2013 Imperial College Computing Student Workshop

ICCSW'13, September 26–27, 2013, London, United Kingdom

Edited by

Andrew V. Jones

Nicholas Ng

Editors

Andrew V. Jones
Department of Computing
180 Queen's Gate, London, SW7 2AZ
United Kingdom
andrewj@doc.ic.ac.uk

Nicholas Ng
Department of Computing
180 Queen's Gate, London SW7 2AZ
United Kingdom
nickng@doc.ic.ac.uk

ACM Classification 1998
A.0. Conference Proceedings

ISBN 978-3-939897-63-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-63-7.

Publication date
September, 2013

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.ICCSW.2013.i

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW

Preface

We are pleased to present the proceedings of the third Imperial College Computing Student Workshop (ICCSW'13), which took place on 26th–27th September 2013 in London, and was hosted by Imperial College London.

ICCSW is an event organised with the "by students, for students" ethos in mind. The organisation work of the workshop was done by a committee formed of Ph.D. students in the Department of Computing, Imperial College London.

The vision for ICCSW is to provide a venue for doctoral students to experience running and participating in an academic workshop, as well as providing a networking opportunity for researchers in similar stages of their career. All of the papers and reviews for ICCSW were written by students; we adopted a unique peer review system in which all authors are also programme committee members for the workshop.

For the workshop's second year, we introduced a highly successful ambassador programme for students in different institutions to promote ICCSW. This invaluable programme helps to increase the scope and prominence of the event both nationally and internationally.

These proceedings contain 19 contributions in various fields from across computer science. This year the workshop received 27 submissions over two tracks: technical papers track, and tools and demonstrations track.

After a rigorous review and selection process, 19 papers were accepted to be presented at ICCSW'13, representing a 70% acceptance rate.

Both days of the workshop featured a keynote from a prominent researcher in computer science; we were truly grateful to have Turing Award winner, Sir Tony Hoare and Google's Director of Research, Peter Norvig, as our keynote speakers.

The talks were entitled:

- Laws of Programming with Concurrency, by Tony Hoare (Microsoft Research); and
- Building Better Online Courses, by Peter Norvig (Google Inc.)

On behalf of the organising committee, we wish to thank all authors, accepted or not, and our ambassadors, who acted as reviewers in our unique peer review process. Furthermore, we also wish to thank our sponsors: Imperial College London, who provided us with more than just financial support; Google, our platinum-level sponsor, who have supported ICCSW since its inception in 2011; Facebook for their gold-level sponsorship; and ARM and HP for their bronze-level sponsorship. Without the support of our sponsors, ICCSW'13 would not have been possible.

Andrew V. Jones and Nicholas Ng
ICCSW'13 Editors

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organisation

Organising Committee

Feryal Mehraban Pour Behbani	Imperial College London
Ali Ghoroghi	Imperial College London
Marcel Chris Guenther	Imperial College London
Petr Hošek	Imperial College London
Andrew V. Jones	Imperial College London
Roman Kolcun	Imperial College London
Rumyana Neykova	Imperial College London
Nicholas Ng	Imperial College London
Marily Nika	Imperial College London
Claudia Schulz	Imperial College London
Călin-Rares Turliuç	Imperial College London
Zhongliu Xie	Imperial College London
	Imperial College London ACM Student Chapter `http://acm.doc.ic.ac.uk/`

Ambassadors

Theofanis Apostolopoulos	King's College London
Khulood Alyahya	University of Birmingham
Reza Asadi	Northeastern University
Matthew Forshaw	Newcastle University
Sarah Gaggl	TU Dresden
Cristian Gratie	University Politehnica of Bucharest
Evgenios Hadjisoteriou	University of Cyprus
Jesus Omana Iglesias	University College Dublin
Nuo Li	University of Nottingham
Nuno Dias Martins	University of Lisbon
Andrei Melnik	University of Osnabruck
Artur Meski	University of Lodz
Amin Mobasheri	Delft University of Technology
Amin Mobasheri	Heidelberg University
Qais Noorshams	Karlsruhe Institute of Technology
Ali Oghabian	University of Helsinki
Fatemeh Pakpour	University of Manchester
Marco Paolieri	University of Florence
Alireza Pourranjbar	University of Edinburgh
William Sonnex	University of Cambridge
Nasim Souri	Bournemouth University
Max Tschaikowski	LMU Munich
Hu Xu	University of Dundee
Marcelo Serrano Zanetti	ETH Zurich
Huanzhou Zhu	University of Warwick

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Khulood Alyahya	University of Birmingham
Theofanis Apostolopoulos	King's College London
Reza Asadi	Northeastern University
Leonardo Bartoloni	University of Pisa
Andrea Canciani	University of Pisa
Raul Castro Fernandez	Imperial College London
Leo de Penning	TNO Behaviour and Societal Sciences
Stefan Ellmauthaler	Leipzig University
Valentina Fedorova	Royal Holloway, University of London
Matthew Forshaw	Newcastle University
Manoel França	City University London
Nentawe Gurumdimma	University of Warwick
Sardar Jaf	University of Manchester
Maria Kechagia	Athens University of Economics and Business
Xin Lu	University of Warwick
Fanlin Meng	University of Manchester
Amin Mobasheri	Heidelberg University
Davide Morelli	University of Pisa
Ali Oghabian	University of Helsinki
Jesus Omana Iglesias	University College Dublin
Mert Ozkaya	City University London
Marco Paolieri	University of Florence
Alan Perotti	University of Turin
Quoc-Sang Phan	Queen Mary, University of London
Łukasz Rogowski	University of Lodz
Petch Sajjacholapunt	University of Warwick
Lei Shi	University of Warwick
Wilson Tan	University of Warwick
Ionuţ Ţuţu	Royal Holloway, University of London
Bas van Gijzel	University of Nottingham
Tim Wood	Imperial College London
Hu Xu	University of Dundee
Ali Mustafa Zaidi	University of Cambridge
Marcelo Serrano Zanetti	ETH Zurich
Huanzhou Zhu	University of Warwick

Supporters and Sponsors

Supporting Scientific Institutions

Imperial College London
http://www.imperial.ac.uk/

Platinum Sponsors

Google Inc.
http://www.google.com/

Gold Sponsors

Facebook Inc.
http://www.facebook.com/

Bronze Sponsors

ARM Holdings, plc.
http://www.arm.com/

Hewlett-Packard Company
http://www.hp.com/

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW

Laws of programming with concurrency

Tony Hoare

Microsoft Research

Abstract

The algebraic laws for programming with concurrency are as simple as (and very similar to) the familiar laws of arithmetic. Yet they are stronger for reasoning about the properties of programs than the axioms of Hoare Logic and the rules of an operational semantics put together.

1998 ACM Subject Classification F.1.2 Parallelism and concurrency

Keywords and phrases Concurrency,Programming

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.1

Category Invited Talk

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 1–1

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Building Better Online Courses

Peter Norvig

Google Inc.

Abstract

We now have many choices in designing a course, whether it is in the classroom, online, or a hybrid. This talk will cover some of the mechanics of running an online course, including the factors involved in building a community. And we will discuss whether building a course is like building software: in the early days, software was crafted by individuals, but over time we established processes that enabled large groups to build much larger systems. Today, courses are still crafted by an individual teacher — if we want to build a larger class, serving more students, and more potential paths through the material, do we need a new set of course building processes? How can we assure that our courses will continually improve in quality?

1998 ACM Subject Classification K.3.1 Distance learning

Keywords and phrases Online courses

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.2

Category Invited Talk

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 2–2

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICCSW

A swarm based heuristic for sparse image recovery

Theofanis Apostolopoulos

King's College London, Department of Informatics
Strand, London, WC2R2LS, United Kingdom
theofanis.apostolopoulos@kcl.ac.uk

Abstract

This paper discusses the Compressive Sampling framework as an application for sparse representation (factorization) and recovery of images over an over-complete basis (dictionary). Compressive Sampling is a novel new area which asserts that one can recover images of interest, with much fewer measurements than were originally thought necessary, by searching for the sparsest representation of an image over an over-complete dictionary. This task is achieved by optimizing an objective function that includes two terms: one that measures the image reconstruction error and another that measures the sparsity level. We present and discuss a new swarm based heuristic for sparse image approximation using the Discrete Fourier Transform to enhance its level of sparsity. Our experimental results on reference images demonstrate the good performance of the proposed heuristic over other standard sparse recovery methods (L1-Magic and FOCUSS packages), in a noiseless environment using much fewer measurements. Finally, we discuss possible extensions of the heuristic in noisy environments and weakly sparse images as a realistic improvement with much higher applicability.

1998 ACM Subject Classification I.4.0 Image processing software

Keywords and phrases Compressive Sampling, sparse image recovery, non-linear programming, sparse representation, linear inverse problems

Digital Object Identifier 10.4230/OASIcs.ICCSS.2013.3

1 Introduction

The famous sampling theorem of Shannon-Nyquist has been very important in engineering. Straightforward and precise, it sets forth the number of measurements required to reconstruct any type of signal or image data. However, many real world applications, such as sound, images and video are represented, stored and processed in computers as big files or collections of bits, which has many disadvantages in comparison with small files; they require more storage space, they take longer to transmit and they demand an overwhelming computational cost for processing. For this purpose many signal/image compression techniques have been introduced including the emerging field of Compressed Sensing (CS). Compressive Sampling or Compressed Sensing (CS) is a fairly new area which was previously introduced empirically in the sciences (e.g. by Claerbout-Muir in Seismology) [3, 4, 5, 9]. CS as a cheap and fast sampling and recovery process has attracted considerable research with several new application areas over the past few years. By exploiting the image (sparsity) and the measurements (random samples) structure we are able to recover an image from what was previously considered as highly incomplete and inaccurate (under-sampled) measurements. Following the pioneer theoretical and practical works by Donoho [6], Candes, Romberg and Tao [4, 5, 9, 22] we are able to recover an under-sampled image, with high probability, by solving an ill-posed inverse problem, as a combinatorial optimization problem. Towards this direction, many variants and extensions of CS have been introduced in the literature recently (1000+ papers in the last 8 years) [21]. This paper proposes a new swarm based heuristic

Imperial College Computing Student Workshop 2013 (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 3–10
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for sparse image approximation and representation based on the key mathematical aspects of the CS method. We have already discussed and suggested the basics of this approach in signals [1, 17]. In this paper we aim to preset and extend the heuristic in images which introduces a more complete and realistic application. The heuristic is also compared with other well-known alternative methods in terms of recovery error, samples size and average computation time. The rest of this paper is organised as follows: The next Section presents the sparse image recovery problem. Then, we briefly discuss two well-known methods used for sparse image recovery (Section 3), while the proposed swarm-based iterative method is described in Section 4. Section 5 presents some experimental results of the proposed heuristic and its comparison with the other methods, while the Section 6 provides some conclusions and extensions of the proposed method.

2 Images as sparse representations

In computers, a image can be represented as a two dimensional array of points of the same size as the image. Each of these points is called pixel. Every pixel as a sample from the image and an element of its corresponding matrix represents the spatial irradiance distribution at the corresponding position. In other words, a pixel can be seen as a continuous function f of two variables m, n which correspond to its position of the array/grid (coordinates), while the function's value represents the type of light/color intensity. This light intensity value depends on the standard followed; it can be one value representing the tone of gray (gray level images) or multiple values for colour images, such as RGB and HSI pallets. For example, the corresponding array for a digital 512×512 gray level $2D$ image with 8 bit representation standard (256 colour intensity values) can be defined as [12, 18, 19, 22]:

$$f = \{f(m, n) = z; m, n = 0 : 511, z = 0 : 255\} \tag{1}$$

Sometimes to further enhance the processing steps or operations in an image (and thus its sparsity) we need to apply a so-called Unitary Transform [4, 12, 19, 22]. By this way we change the domain of representation (i.e. image function) from spatial (pixels) to frequency (spectra). In this case the image is represented as a linear combination of basis functions of a linear integral transform. This operation converts an image into one having relatively fewer values significantly different from zero. Obviously, the pursue of the best Transform domain which leads to the sparsest representation highly depends on the trade-off between the computation time and the size of the dictionary basis [12, 18, 19]. In this paper, we will apply the Discrete Fourier Transform (DFT), which uses cosines and sines as basis functions (i.e. $e^{i\omega} = \cos(\omega) + i\sin(\omega)$), to gray level images. The 2D DFT (spectrum) of an image $f(m, n)$ can be defined as [12, 18, 19, 22]:

$$X(u, v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) \exp[-2\pi i(\frac{mu}{M} + \frac{nv}{N})], \tag{2}$$

for $u = 0, 1, \ldots, M-1$ and $v = 0, 1, \ldots, N-1$. The inverse DFT is given by:

$$f(m, n) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} X(u, v) \exp[-2\pi i(\frac{mu}{M} + \frac{nv}{N})], \tag{3}$$

for $m = 0, 1, \ldots, M-1$ and $n = 0, 1, \ldots, N-1$. Usually in images $M = N$, which is also the case for the test images discussed in this paper. Note also that there is one-to-one mapping between the spatial and frequency domain. The 2D DFT maps an $M \times N$ real-valued matrix

$f(m,n)$ on $M \times N$ complex-value matrix $X(u,v)$, while the inverse DFT maps the $X(u,v)$ on $f(m,n)$ [12, 18, 19]. In practice, the DFT is computed using the Fast Fourier Transform (FFT) algorithm, which is nothing but a computationally efficient way of obtaining the DFT coefficients based on the symmetries of the basis (matrix Ψ) [12, 18, 19, 22]. Many natural images have concise representations when expressed in a convenient basis. In CS we use DFT to enhance the sparsity of an image before down-sampling it. In image processing, for simplicity reasons (eg. Histogram of an image), it is very common to treat an $N \times N$ image as a $N := N^2$ vector and samples as a vector on the M frequencies ($M \ll N$); principle we will also adopt here. Let's assume we have a noiseless image $f \in \Re^N$ which we expand in an orthonormal basis (such as a Fourier basis) $\Psi = [\psi_1, \psi_2, \ldots, \psi_N]$ as $X = \sum_{i=1}^{N} f_i \psi_i$. Then the image can be represented as a sparse linear combination of atoms in Φ. In vector format, we have [3, 4, 5, 6, 12, 22]:

$$X_{N\times 1} = \Psi_{N\times N} f_{N\times 1}, \tag{4}$$

where Ψ is a unitary $N \times N$ matrix (basis) with $\psi_1, \ldots, \psi_N$ as columns and X is the vector of frequency coefficients with respect to Ψ. So, the N-point DFT is expressed as an N-by-N matrix multiplication, where f is the original input image and X is the DFT of the image. Then we can sense or collect partial information about X (measurements) as $y_k =< X, \phi_k >, k = 1, 2, ..., M$ or in vector format as [3, 4, 5, 6, 12, 22]:

$$Y_{M\times 1} = C_{M\times N} X_{N\times 1} = \Phi_{M\times N} \Psi_{N\times N} f_{N\times 1} \tag{5}$$

That is, we simply correlate the object we wish to acquire with the waveforms Φ, which is the measurement or sampling operator and Ψ is the sparcifying operator (Fourier transform) [3, 4, 5, 6, 8]. In fact, there is no formal difference between Φ and Ψ. In theory, the former refers to the dictionary of physical spectra and the later refers to the dictionary of image waveforms. In practice, Ψ is a partial Fourier matrix obtained by selecting M rows (i.e. measurements) uniformly at random, using Gaussian distribution, and then re-normalising the columns so that they are unit-normed (See [5, 6, 8, 12, 19, 22]). Note that the random Fourier ensemble is only used as a more realistic application and thus efficient recovery of the original image f still requires a unique sparsest solution. In a nutshell, the key steps of CS are: take the DFT of the desired image to enhance its sparsity, under-sample it (lossy compression) randomly, transmit/store it and then decompress (recover) it by solving an optimisation problem. As we will see in the next section, different recovery methods solve slightly different optimisation problems, though all these approaches serve the same purpose: the sparse recovery of a (compressed) image as a solution to an optimisation problem.

3 Methods for sparse recovery in images

CS is very advantageous in images which are sparse (have only a few non-zero entries) in a known basis provided that the measurements collected are incoherent (i.e. random) [3, 4, 5, 8, 12, 19, 22]. Since we are interested in sparsely representing highly under-sampled images, the linear system describing the measurements in (5) is under-determined and therefore has infinitely many solutions [3, 4, 5, 6]. This instance of an under-determined system of linear equations constitutes a linear inverse problem (LIP) [4, 5, 6, 8]. In this paper we will consider C as a random Fourier ensemble (rows are randomly chosen DFT vectors), in a noiseless environment. Note that randomness of sampling guarantees that we have a linearly independent system of equations and hence a unique solution. We will also restrict our approach to image restoration experiments applied to this problem and not to

general applicability LIPs [4, 5, 8]. We discuss two well-known optimisation principles which are implemented in Matlab packages and have been extensively studied mathematically.

3.1 The L1 Magic

L1 MAGIC is a collection of MATLAB routines, based on standard interior-point methods, for solving optimization programs relevant to Compressive Sampling [5, 7]. In the case of the sparse image (noiseless) recovery problem the L1 Magic solves the TV minimisation problem with equality constraints which is a Second Order Cone Programming (SOCP) [4, 5, 7, 22]:

$$\min\ TV(X)\ \text{ s.t. }\ CX = Y, \tag{6}$$

where C is the Sampling/Sensing matrix (the under-sampling Fourier operator F_u), Y is the measurements vector, while TV stands for the Total Variation, which is the sum of magnitudes of the discrete gradient $D_{ij}X$ at every point/pixel x_{ij} of a FFT image X with i representing the rows and j representing the columns (assume sparsity in gradients) [4, 5, 7, 22]:

$$TV(X) := \sum_{ij}^{N-1} \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2} = \sum_{ij} \|D_{ij}X\|_2 \tag{7}$$

The L1 Magic uses a log-barrier method to solve the SOCP in (6). It initially transforms it into a series of linearly constrained problems and then solves them by forming a a series of quadratic approximations (i.e. a Newtonian iteration step which proceeds by minimizing each of these systems of equations) [7].

3.2 The FOCUSS

FOCUSS package, which stands for FOCal Under-determined System Solver, is an algorithm designed to obtain sub-optimally sparse solutions to linear inverse problems in relatively noise-free environments [14, 20]. It is an affine-scaling transformation interior point optimisation algorithm which is based on conjugate gradient factorisation for finding sparse solutions of the following concave function [14, 20]:

$$\hat{X} = \ \text{args}\ \min_X \frac{1}{2}\|Y - CX\|^2 + \lambda d_p(X), \tag{8}$$

where C is the Sampling/Sensing matrix, Y is the measurements vector, $0 < \lambda < 1$ is a regularisation parameter. This reflects the trade-off between the sparse residual $\|Y - C\hat{X}\|$ and the sparse source vector estimate $\hat{X}$ and depends on the compression rate of the sampled FFT image X. The quantity $d_p(X)$ corresponds to the following norm [14, 20]:

$$d_p(X) = \|X\|_{l_p} = \sum_{i,j} \|x_{ij}\|^p, \tag{9}$$

for $0 < p \leq 1$ which enforces sparse solutions to the problem.

4 Research Approach

A simple technique for recovering an image of interest X from partial measurements Y is to find a solution from an infinite set with the minimum sparsest norm [3, 4, 5, 6, 8, 9, 12, 19]:

$$\min\ \|X\|_{l_0}\ \text{ s.t. }\ Y = CX \tag{10}$$

where, the norm $\|.\|_{l_0}$ counts the non-zero elements of the vector X and thus $\|X\|_{l_0} = S$ for a S-sparse image (S non-zero entries). As X represents the partial Fourier measurements the image can be reconstructed as $f = \Psi X$. A common approach to overcome the difficulties of the combinatorial search required for solving (10) would be to replace it by its convex relaxation and particularly by substituting the l_1 norm for the l_0 pseudo-norm (For details see [3, 4, 5, 6, 9]). In this paper, we will follow a different approach which introduces an efficient way to approximate the l_0 by the following smoother, continuous and easier to differentiate Laplace function [1, 10, 13, 15, 16, 17]:

$$\|X\|_{l_0} \approx f(|X|,\sigma) = \sum_{i=1}^{N} 1 - f(|x_i|,\sigma) = N - \sum_{i=1}^{N} \exp(-\frac{|x_i|^2}{2\sigma^2}), \tag{11}$$

where x_i is the i-th element of vector X of length N and σ is a sequence index parameter. Among the advantages of this approach are the robustness of the l_0 norm to noisy samples, the number of measurements required, which is much smaller than the ones required by its convex analog (l_1 norm), and less restrictions in the design of Sensing matrices C. The problem in (10) is now reformed as an unconstrained optimisation problem:

$$\min \quad f(|X|,\sigma) = (M - \sum_{i=1}^{M} \exp(-\frac{(y_i - c_i x_i)^2}{2\sigma^2})) \tag{12}$$

where x_i and y_i are the i-th elements of vectors $X \in C^N$ and $Y \in C^N$ respectively, while c_i represents the i-th row of Sensing matrix $C \in C^{M \times N}$ with $M \ll N$. The purpose is to minimise both the objective function in (12) and the parameter σ. The value of this parameter represents the tradeoff between accuracy and smoothness of the approximation. The smaller the σ, the better the approximation, while the larger the σ, the smoother the approximation. In fact, the functional in (11) interpolates the function space between l_1 and l_0 across $\sigma \in [0, \infty)$ in the same manner as does l_p norm for $p \in [0, 1]$ (i.e. $f(1, \sigma) = 1$ and $\sigma \to \infty$ admits $f(|X|, \sigma) \to \|X\|$). This approach has been successfully used for similar recovery problems and proven to yield a unique and sparse solution similar to the approach yielded by the l_0 quasi-norm (See [10, 13, 15, 16] for details and extensive results).

4.1 The Heuristic

The heuristic is an iterative stochastic swarm-based method for finding the global minimum of a non-convex, unconstrained continuous function in (12). The steps of the heuristic are:

Proposed l_0-norm based Heuristic:

```
Problem: Determine X ∈ C^N s.t. CX = Y.
Inputs: σ, C, Y, Iterations, Agents, Sparsity S, f(|X|,σ), Basis Ψ.
Outputs: best value f_*(|X_*|,σ), best sparse vector X_*, image f_*.
Main steps of the swarm based Heuristic:
Initial solution for every swarm i: X_i^(0) = ((C^T C)^-1 C^T Y) + R^(0) × ||C^T Y||_∞
Set σ_i^(0) = ||Y − X_i^(0)||_∞ for every swarm i
  While (t < Iterations)
    For i := Agents/Swarms to 25 Do
          Evaluate f(|X|,σ) for every X_i^(t)
          Find current best X_*^(t) so as min f(|X|,σ)
          Set X_*^(t) = X_i'^(t) (keep the best i'th solution)
          Check X_*^(t) entries for feasibility
          Set all but S largest entries of X_*^(t) to zero
```

```
            Generate new solutions for all the other agents using (14)
    End For loop;
  Set σ^(t+1) = σ^(t) × 0.6
  End While loop;
Reconstruct image f* = Ψ^-1 X* (IFFT of X* ∈ C^N to derive f* ∈ ℜ^N).
Display the recovered image f*,Calculate the time and error recovery.
```

Initially the heuristic is initialised with a population of 25 agents each of which carrying a slightly different solution and σ parameter. A variation of pseudo-inverse $(((C^TC)^{-1}C^TY) + R^{(0)} \times \|C^TY\|_\infty)$ is chosen as an estimate of the initial sparse solution in (10), which will be further improved through the iterations of the heuristic. $X_i^{(t)}$ is the current solution vector for agent i at time t, while R is a vector of randomly generated values between 0 and 1, using the Normal distribution. Note that $\|.\|_\infty$ is the infinity or Chebychev norm, which is defined as $\|L\|_\infty = \max \|l_1\|, \ldots, \|l_N\|$ for a vector $L = [l_1, \ldots, l_N]$ in a finite dimensional coordinate space. Note that, at each iteration the current best solution $X_*^{(t)}$ is chosen after being corrected for sparsity and feasibility (be within the ranges of the original transformed image). Then a new solution is created for each remaining particle which is updated based on the following rule:

$$X_i^{(t)} = 2 \times R^{(t)} \times X_i^{(t-1)} + (1 - R^{(t)}) \times \sigma^{2L} \times L, \tag{13}$$

where, R is a vector of small random numbers, different for every swarm i, (t) is the current iteration, $X_i^{(t)}$ and $X_i^{(t-1)}$ is the current and the previously generated solution vector of swarm i and L is the infinity norm $\|C^T(CX_i^{(t-1)} - Y)\|_\infty$. The σ value is initially assigned to the maximum value between the samples vector and the sampled pseudo-inverse and then it is gradually decreased at each iteration. This assignment was chosen experimentally based on the nature of the initial vector. Note that due to the randomness in each step of the heuristic, there is no mathematical guarantee of achieving a global minimum as does its convex l_1 analogue. However, the local minimum found by solving the non-convex problem in (12) typically allows for accurate and successful recovery even at much higher under-sampling rates where linear optimisation fails (See Section (5) for details).

5 Simulations and Results

All the numerical experiments were performed on an Intel Core i5 CPU (3.20 GHz) with 3 GB RAM, using Matlab R2012b under MS Windows XP Pro. We have tested the performance of the heuristic as a sparse recovery method in two 256×256 images which have been extensively used for testing purposes, namely Shepp-Logan Phantom and Circles (See [2, 9, 13, 19, 21]). The result of the experiments is shown in Figure (1) which presents the original images, the Sampling pattern (i.e. number of lines through origin), the Back-projection (C^TY), which represents direct recovery from partial measurements, and the recovered image (estimate) using the methods L1 Magic, FOCUSS and the Heuristic. The performance of the heuristic is compared with the other methods in terms of recovery error (RE) as a metric to evaluate the recovered image quality. The recovery error was calculated as $\text{RE} = (\|\hat{X} - X\|_{l_2})/(\|X\|_{l_2})$, where $\hat{X}$ and X is the recovered and original image respectively, while the CPU cycles were used as a rough estimation of execution time for all the methods. The average time for the phantom image recovery was 468.27 ($\sim$ 10 mins) for L1 Magic with 15 Log-barier iterations, 322.85 ($\sim$ 6 mins) for FOCUSS with 15 iterations, $\lambda = 2.0e-3$ and $p = 0.5$, and 120 ($\sim$ 3 mins) for the heuristic with 23 iterations and 25 agents. The average time for the circle image recovery was 448 ($\sim$ 7 mins) for L1 Magic, 290 ($\sim$ 5 mins) for FOCUSS and 86 ($\sim$ 2

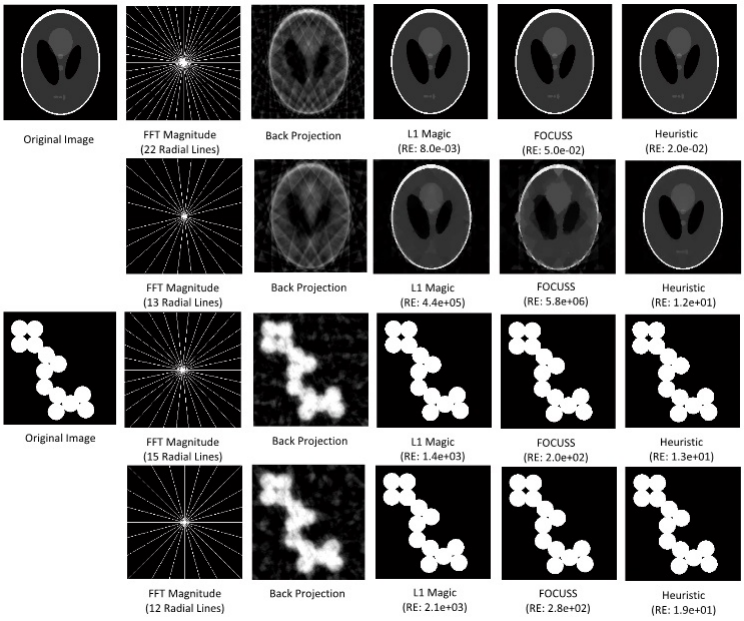


Figure 1 Image recovery experiments for L1 Magic, FOCUSS and Heuristic.

mins) for the heuristic, using the same parameters as previously. Notice that a few frequency coefficients (magnitudes) can capture most of the image energy, as most of such images are highly compressible. Notice also that the performance of the heuristic (in accordance with all the other methods) is increasing as the number of measurements increases and deteriorates as the sparsity of images decreases, which is expected as fewer measurements cause loss of image quality and thus loss of substantial information (i.e. aliasing in the reconstruction). However, the heuristic is found to have significantly better performance with smaller run times than the other recovery methods, particularly for much under-sampled data (less than 15 radial lines), while the difference between the recovered and the original image is hardly noticeable in some cases (particularly for more than 15 radial lines).

6 Conclusions

In this paper the performance of the proposed method for sparse image recovery was studied and compared with other methods. The heuristic essentially helps in faster and quicker sparse recovery of the test images by solving a non-convex unconstrained optimization problem with complex values, resulting in decreasing the requirement of the number of measurements needed by other alternative sparse recovery algorithms. It is expected that the performance of the heuristic, especially in noisy environments, can be improved by assigning weights to the objective function as an efficient way to improve the search direction. This approach has been proven to be useful and helps in better recovery for similar recovery problems and methods using the l_1, l_2 and l_0 norms (See [9, 11, 13]). Another possible direction is to investigate if we could design Sensing matrices which do not follow the Gaussian and Bernoulli distributions, or the random Fourier ensemble. These are the only distributions used to efficiently recover strictly sparse images from corrupted measurements (using the l_1 norm) as they satisfy the properties of UUP and RIP (i.e. mutual coherence between the Basis Ψ and the Sensing Φ matrices, for details See [4, 5, 8]). However, images of practical

interest are generally weakly sparse or compressive, in essence that their sorted magnitudes in a known basis usually decay exponentially, and thus further experimentation may yield to reveal an even better recovery for weakly sparse images and signals (See [12, 19]).

Acknowledgements The author would like to thank his supervisor, Dr. Tomasz Radzik, for his insight and constructive comments and the anonymous reviewers for their suggestions.

References

1 T. Apostolopoulos. A heuristic for sparse signal reconstruction. In *ICCSW 2012*, volume 28 of *OASIcs*, pages 8–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.
2 Calibrated Imaging Lab at Carnegie Mellon University. Collection of test images. http://www.cs.cmu.edu/afs/cs/project/cil/ftp/html/v-images.html.
3 R. Baraniuk. Compressive sensing. *IEEE Signal Processing Magazine*, pages 118–120, 2007.
4 E. J. Candès. Compressive sampling. *Proceedings of the International Congress of Mathematicians, Madrid, Spain*, 2006.
5 E. J. Candès. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans on Information Theory*, 52(2):489–509, 2006.
6 D. Donoho. Compressed sensing. *IEEE Trans on Information Theory*, 52(4):1289–1306, 2006.
7 E. J. Candès et al. L1-magic: Recovery of sparse signals via convex programming. http://users.ece.gatech.edu/ justin/l1magic/downloads/l1magic.pdf.
8 E. J. Candès et al. Sparsity and incoherence in compressive sampling. *Inverse Problems*, 23(3):969–985, June 2007.
9 E.J. Candès et al. Enhancing sparsity by reweighted l_1 minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, December 2004.
10 H. Mohimani et al. A fast approach for overcomplete sparse decomposition based on smoothed l_0 norm. *IEEE Trans on signal processing*, 57(1):289–301, November 2009.
11 J. K. Pant et al. Reconstruction of sparse signals by minimizing a re-weighted approximate l_0-norm in the null space of the measurement matrix. *Circuits and Systems,53rd IEEE International Midwest Symposium*, pages 430–433, August 2010.
12 J. L. Starck et al. *Sparse Image and Signal Processing; Wavelets, Curvelets, Morphological Diversity*. Cambridge University Press, UK, 2010.
13 J. Trzasko et al. Highly undersampled magnetic resonance image reconstruction via homotopic l_0-minimisation. *IEEE Trans. on Medical Imaging*, 28(1):106–121, January 2009.
14 K. Kreutz-Delgado et al. Dictionary learning algorithms for sparse representation. *Neural Computation*, 15(2):349–396, February 2003.
15 P. Huber et al. *Robust Statistics*. Wiley, 2 edition, 2009.
16 S. Ashkiani et al. Error correction via smoothed l_0-norm recovery. *IEEE Statistical Signal Processing Workshop (SSP)*, pages 289–292, June 2011.
17 T. Apostolopoulos et al. A swarm based method for sparse signal recovery. In *ARSR/SWICOM 2013, Luton*, pages 1–5, 2013.
18 B. Jaehne. *Digital Image Processing*. Springer, 2002.
19 S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 3 edition, 2009.
20 J. F. Murray. Focuss website. http://dsp.ucsd.edu/ jfmurray/software.htm.
21 Compressive Sensing Resources. Rice university. http://dsp.rice.edu/cs.
22 J. Romberg. Imaging via compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):14–20, March 2008.

Scalable and Fault-tolerant Stateful Stream Processing*

Raul Castro Fernandez[1], Matteo Migliavacca[2], Evangelia Kalyvianaki[3], and Peter Pietzuch[1]

1 **Dept. of Computing, Imperial College London**
rc3011@doc.ic.ac.uk, prp@doc.ic.ac.uk
2 **Dept. of Computing, University of Kent**
mm53@kent.ac.uk
3 **School of Informatics, City University London**
evangelia.kalyvianaki@city.ac.uk

Abstract

As users of "big data" applications expect fresh results, we witness a new breed of stream processing systems (SPS) that are designed to scale to large numbers of cloud-hosted machines. Such systems face new challenges: (i) to benefit from the "pay-as-you-go" model of cloud computing, they must scale out on demand, acquiring additional virtual machines (VMs) and parallelising operators when the workload increases; (ii) failures are common with deployments on hundreds of VMs—systems must be fault-tolerant with fast recovery times, yet low per-machine overheads. An open question is how to achieve these two goals when stream queries include stateful operators, which must be scaled out and recovered without affecting query results.

Our key idea is to expose internal operator state explicitly to the SPS through a set of state management primitives. Based on them, we describe an integrated approach for dynamic scale out and recovery of stateful operators. Externalised operator state is checkpointed periodically by the SPS and backed up to upstream VMs. The SPS identifies individual operator bottlenecks and automatically scales them out by allocating new VMs and partitioning the checkpointed state. At any point, failed operators are recovered by restoring checkpointed state on a new VM and replaying unprocessed tuples. We evaluate this approach with the Linear Road Benchmark on the Amazon EC2 cloud platform and show that it can scale automatically to a load factor of L=350 with 50 VMs, while recovering quickly from failures.

1998 ACM Subject Classification H2.4 Database Systems. Systems

Keywords and phrases Stateful stream processing, scalability, fault tolerance

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.11

1 Introduction

In many domains, "big data" applications [2], which process large volumes of data, must provide users with fresh, low latency results. For example, web companies such as Facebook and LinkedIn execute daily data mining queries to analyse their latest web logs [8]; online marketplace providers such as eBay and BetFair run sophisticated fraud detection algorithms on real-time trading activity [7]; and scientific experiments require on-the-fly processing of data.

* A longer version of this paper appeared in the proceedings of ACM International Conference on Management of Data (SIGMOD) [4].

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 11–18

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Therefore *stream processing systems* (SPSs) have evolved from cluster-based systems, deployed on a few dozen machines [1], to extremely scalable architectures for big data processing, spanning hundreds of servers. Scalable SPSs such as Apache S4 [6] and Twitter Storm [12] parallelise the execution of stream queries to exploit *intra-query parallelism*. By scaling out partitioned query operators horizontally, they can support high input stream rates and queries with computationally demanding operators.

While mechanisms for scale out [10, 9] and fault tolerance [13, 11, 15] in stream processing have received considerable attention in the past, it remains an open question *how SPSs can scale out while remaining fault tolerant when queries contain* ***stateful operators***. Especially with recently popular stream processing models [6, 12] that treat operators as black boxes in a data flow graph, users rely on operators that have large amounts of state, which potentially depends on the complete history of previously processed tuples [3]. This is in contrast to, for example, window-based relational stream operators [1], in which state typically only depends on a recent finite set of tuples.

We make the observation that both scale out and failure recovery affect operator state, and therefore can be solved more efficiently using a single integrated approach. Our key idea is to externalise internal operator state so that the SPS can perform explicit operator **state management**. We then define a set of primitives for state management that allow the SPS to *checkpoint*, *backup*, *restore* and *partition* operator state. Based on these primitives, we describe an **integrated approach for scale out and recovery** of stateful operators in an SPS.

We evaluate how our approach scales out queries as part of a prototype SPS using closed and open loop workloads. We report the performance of the Linear Road Benchmark [3] on the Amazon EC2 cloud platform.

In summary, the paper makes the following contributions:

1. a description of operator state and management primitives to be used by an SPS;
2. an integrated approach for automatically scaling out bottleneck operators and recovery of failed operators based on managed operator state;
3. an experimental evaluation on a public cloud, showing that this approach can parallelise complex queries to a large number of VMs, while being resilient to failures.

Next we analyse the problem; §3 presents our state management technique; based on this, we introduce the integrated approach for scale out and recovery (§4); §5 provides experimental results; and we finish with conclusions (§6). For related work and further details on this paper we refer the reader to [4].

2 Problem Statement

We want to enable the deployment of SPSs on *infrastructure-as-a-service* (IaaS) clouds, such as Amazon EC2 and Rackspace, across hundreds of VMs. An SPS in a cloud setting must support the *automated* deployment and management of stateful streaming queries. In particular, this requires (i) the exploitation of *intra-query parallelism* to scale processing across VMs; (ii) the masking of *failures* for continuous processing; and (iii) adaptation to a *VM model*.

Stateful operators. Existing systems typically assume that operators are either stateless [6] or that state can be ignored when e.g. recovering operators [12]. While this simplifies the architecture of the SPS, it puts a considerable burden on developers when they need scalable and fault-tolerant stateful operators.

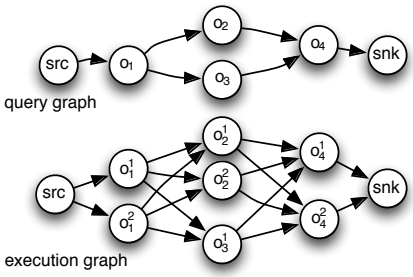


Figure 1 Example of query and execution graphs.

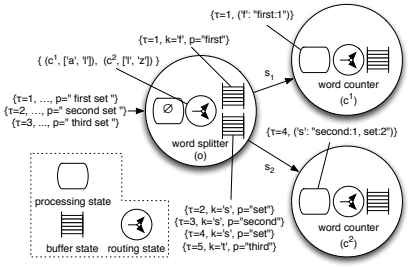


Figure 2 Different types of state in a stateful query for counting word frequencies.

Intra-query parallelism. Decisions about parallelising operators can occur *statically*—at query deployment time—or dynamically—at runtime. Static scale out requires knowledge of resource requirements of operators, which depend on stream rates and data distributions, and are typically estimated by cost models [14]. Therefore dynamic scale out is preferable in a cloud setting because the SPS can adapt to changes in the workload, observing resource consumption and VM performance.

Fault tolerance. Previous studies have shown that a substantial fraction of machines in large data centres develop faults during operation [5]. We assume a typical failure model, in which machine and network failures are modeled as independent, random crash-stop failures. Similar to other cloud-deployed applications, an SPS must be fault tolerant and cope with regular failures.

3 State Management

System Model

Data Model. A *stream* s is an infinite series of tuples $t \in s$. A *tuple* $t = (\tau, k, p)$ has a logical timestamp τ, a key field k and a payload p. The timestamp $\tau \in \mathbb{N}^+$ is assigned by a monotonically increasing *logical clock* of an operator when a tuple is created in a stream. Tuples in a stream are ordered according to their timestamps. Keys are not unique and used to partition tuples. They can be computed as a hash based on the payload.

Operator model. Tuples are processed by operators. An operator o takes n *input streams*, I_o, processes their tuples and produces one or more *output streams*, O_o.

An *operator function* f_o defines the processing of operator o on input tuples: $f_o : (I_o, \overline{\tau_o}, \theta_o, \overline{\sigma_o}) \rightarrow (O_o, \overline{\tau_o}, \theta_o, \overline{\sigma_o})$. A *stateful* operator has access to state θ_o, which is updated after processing. We assume that operators are deterministic and do not have other, externally visible side-effects. The timestamp $\overline{\sigma_o}$ specifies the oldest tuples that affected the state θ_o, i.e. the state depends only on tuples with timestamps $\overline{\sigma_{oi}} \leq \overline{\tau}_i \leq \overline{\tau_{oi}}$ for each input stream s_i.

Query model. As shown at the top of F. 1, a query is specified as a directed acyclic *query graph* $q = (\mathcal{O}, \mathcal{S})$ where $\mathcal{O}$ is the set of operators and $\mathcal{S}$ is the set of streams.

Query execution. A query is deployed on a set of *nodes*. A node can host multiple operators but, without loss of generality, we assume one operator per node. We distinguish between the logical representation of a query, in terms of its query graph, and its physical realisation, as shown at the bottom of F. 1. In the physical *execution graph* $\bar{q}$, an operator o may be parallelised into a set of *partitioned* operators $o^1 \ldots o^\pi$.

State Definition

The state of a query consists of the *operator state* of each query operator. We divide the operator state into *processing state*, *buffer state* and *routing state*, as illustrated in F. 2, which we use as a running example below.

Processing state. Output tuples from stateful operators depend on input tuples and the history of past tuples. Operators typically maintain an internal summary of this history of input tuples, which we term the operator's *processing state*. The current processing state θ_o of an operator o was computed from all past tuples with $\overline{\sigma_{oi}} \leq \overline{\tau}_i \leq \overline{\tau_{oi}} : s_i \in I_o$.

Exposing the processing state to the SPS has several reasons: (i) it enables the SPS to recover stateful operators more efficiently after failure. Instead of re-processing all tuples in the range $\overline{\sigma_{oi}} \leq \overline{\tau}_i \leq \overline{\tau_{oi}}$, recreating the processing state, the SPS can restore the state directly from a state checkpoint, and (ii) it allows the SPS to redistribute processing state across a set of new partitioned operators to support scale out.

In F. 2, we give an example of processing state for the word frequency operators.The upstream word split operator sends the word "first" to the word count operator c^1 at $\tau = 1$, resulting in the processing state $\theta_{c^1} = \{('f', \text{"first:1"})\}$ and timestamp $\overline{\tau_{c^1}} = (1)$. The words "set", "second" and "set" are processed by c^2, instead, which at $\overline{\tau_{c^2}} = (4)$ holds processing state $\theta_{c^2} = \{('s', \text{"second:1, set:2"})\}$.

Buffer state. An SPS typically interposes *output buffers* between operators, which buffer tuples before sending them to downstream operators (see F. 2). Buffers compensate for transient fluctuations of stream rates and network capacity.

Tuples in output buffers contribute to the query state managed by the SPS: (i) output buffers store tuples that have not yet been processed by downstream operators and therefore must be re-processed after failure; (ii) after dynamic operator scale out, tuples in output buffers must be dispatched to the correct partitioned downstream operator.

Routing state. An operator o in the query graph may correspond to multiple partitioned operators $o^1, \ldots, o^\pi$ in the execution graph. An upstream operator u has to decide to which partitioned operator o^i to route a tuple. Since the partitioning can change dynamically, an operator has explicit *routing state*, which must be restored after failure.

Operations

The above operator state can be manipulated by the SPS through a set of state management primitives.

Checkpoint state. The SPS can obtain a representation of the processing state θ_o and the buffer state β_o of an operator o in the form of a *checkpoint*. This is taken by the function checkpoint-state$(o) \rightarrow (\theta_o, \overline{\tau_o}, \beta_o)$. It obtains the processing state θ_o safely by calling the user-implemented function get-processing-state(), which also returns the timestamp $\overline{\tau_o}$ of the most recent tuples in the streams from the upstream operators that affected the state checkpoint. This permits the SPS to discard tuples with older timestamps, which are duplicates, during replay (see below).

The function checkpoint-state is executed asynchronously and triggered every *checkpointing interval* c, or after a user-defined event, e.g. when the state has changed significantly.

Backup state. The operator state, as returned by checkpoint-state, can be backed up to an upstream operator in anticipation of a restore or partition operation. After the operator state was backed up, already processed tuples from output buffers in upstream operators can be discarded because they are no longer required for failure recovery.

Restore state. Backed up operator state is restored to another operator to recover a

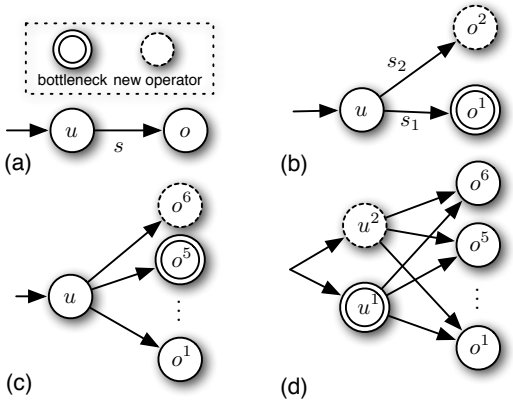


Figure 3 Example of scale out of stateful operators.

failed operator or to redistribute state across partitioned operators. A function takes the state to restore to operator o. It then initialises the processing state using a user-defined function and also assigns the buffer and routing states.

After the state was restored from a checkpoint, unprocessed tuples in the output buffer from an upstream operator are replayed to bring the operator o's processing state up-to-date. Before operator o emits new tuples, it resets its logical clock to the timestamp τ from the restored checkpoint so that downstream operators can detect and discard duplicate tuples.

Partition state. When a stateful operator scales out, its processing state must be split across the new partitioned operators. This is done by repartitioning the key space of the tuples processed by the operator (i.e. by doing consistent hashing). In addition, the routing state of its upstream operators must be updated to account for the new partitioned operators. Finally, the buffer state of the upstream operators is partitioned to ensure that unprocessed tuples are dispatched to the correct partition.

4 Scale Out and Fault Tolerance

Using the above state management primitives, we present our integrated approach for stateful operator scale out and recovery. We discuss our scaling strategy and fault tolerance, before describing our fault-tolerant scale out algorithm.

To scale out queries at runtime, the SPS partitions operators on-demand in response to bottleneck operators. Bottleneck operators prevent the system from increasing processing throughput. After scaling out a bottleneck operator, its processing load is shared among a set of new partitioned operators, thus increasing available resources to the SPS. Our scale out mechanism partitions operator state and streams without violating query semantics.

We give an example of operator scale out in F. 3, which shows four versions of an execution graph during scale out. When first deployed (F. 3a), the execution graph has one operator for each (logical) operator in the query graph. An operator o is connected through stream s to an upstream operator u. We assume that operator o is the bottleneck operator. F. 3b shows how the upstream operator u can partition its output streams into two streams. The two partitioned operators, o^1 and o^2, share the processing load and alleviate the bottleneck condition. In the same way, additional operators can be added to the execution graph for further scale out (F. 3c). When the upstream operator u becomes the new bottleneck (F. 3d), it is also partitioned and its output streams are replicated.

Even in the absence of bottlenecks, if a VM hosting a stateful operator fails, the SPS must replace it with an operator on a new VM. In our approach, overload and failure are

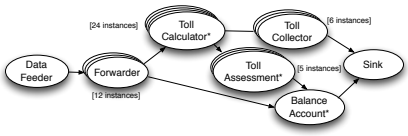


Figure 4 Query for the Linear Road Benchmark.

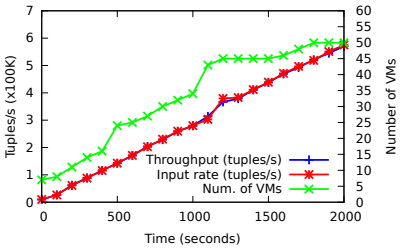


Figure 5 Dynamic scale out for the LRB workload with L=350 (closed loop workload).

handled in the same fashion. Operator recovery becomes a special case of scale out, in which a failed operator is scaled out to a parallelisation level of 1. This means that the SPS does not require a sophisticated failure detector to distinguish between the two cases but instead scales out an operator when it has become unresponsive.

5 Evaluation

The goals of our experimental evaluation are to investigate:
(i) the **effectiveness** of our **stateful operator scale out** approach for a *closed loop* workload.
(ii) the **recovery time** of the **stateful recovery** mechanism for a windowed word frequency query.
(iii) the **impact** of our **state management** approach on tuple processing latency.

The experiments are conducted using an experimental stream processing system implemented in Java. We deploy it on Amazon EC2 across 60 VMs.

Dynamic Scale Out

We first evaluate the effectiveness of our scale out approach when adapting to an increasing workload, i.e. when the SPS has to scale out to match an increasing input stream rate without tuple loss. The workload is the Linear Road Benchmark (LRB) (see [3] for details).

Our LRB query implementation consists of 7 operators, as shown in F. 4. We deploy the LRB query on Amazon EC2. Our deployment achieves a maximum L-rating of L=350 with 50 VMs. After that, the source and sink become the bottleneck, handling a maximum of 600,000 *tuples/s* due to serialisation overheads. The partitioned execution graph of the LRB is as shown in F. 4. We observe that the SPS maintains the required result throughput for the input rate, requesting additional VMs as needed. At times t=475 and t=1016, multiple operators are scaled out in close succession because bottlenecks appear in two operators simultaneously.

F. 6 shows processing latencies of output tuples, as a metric for the performance experienced by the query. The 99^{th} and 95^{th} percentiles of the latency are 1459 *ms* and 700 *ms*, respectively; the median is 153 *ms*, which are all below the LRB target of 5 *s*. This confirms that our maximum L-rating is indeed due to the limited source and sink capacities.

Failure Recovery

To evaluate failure recovery, we first compare recovery time against other fault tolerance approaches, *upstream backup* (UB) and *source replay* (SR). UB buffers tuples in each operator

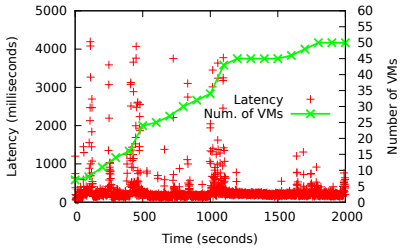


Figure 6 Processing latency for LRB workload.

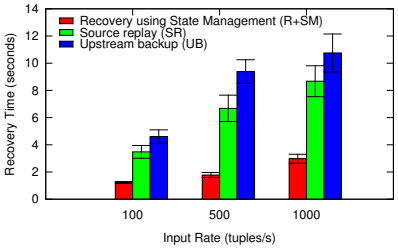


Figure 7 Recovery time for different fault tolerance mechanisms.

and re-processes them to recover operator state. SR is a variant of UB, in which tuples are only buffered and replayed by the source [12]. We use a query that counts word frequencies over a 30 *s* window.

We observe the recovery times for the three approaches. For R+SM, we set the checkpointing interval *c* to 5 *s*. During the experiment, we fail the VM and measure the time to recover (i.e. until the complete operator state was restored).

F. 7 shows results averaged over 10 runs for different input rates. SR achieves slightly faster recovery than UB because of the short length of the operator pipeline and the fact that it stops the generation of new tuples during the recovery phase. R+SM achieves lower recovery times than both UB and SR. Due to the state checkpoints, it re-processes fewer tuples to recover the stateful operator.

In F. 8, we show the change in recovery time as a function of the checkpointing interval for different input rates. Recovery time increases with longer checkpointing intervals because more tuples are replayed. Tuple buffering is the main factor determining recovery time, which is why recovery time increases considerably with higher rates. While frequent checkpointing incurs overhead, it reduces recovery time, even for high rates.

State Management Overhead

The overhead on processing throughput could not be observed, so we measure its effect on tuple processing latency.

We synthetically vary the state size (in this case a dictionary) between *small* (10^2 entries; ≈2 *Kb*), *medium* (10^4 entries; ≈200 *Kb*) and *large* (10^5 entries; ≈2 *Mb*).

F. 9 shows that the 95$^{\text{th}}$ percentile of tuple processing latencies increases with state size. For large state sizes, checkpointing takes longer and occupies more CPU time, which is unavailable for tuple processing. Higher input rates increase the load on the operator, resulting in less headroom for the checkpointing process. For input rates of 100 and 500 *tuples/s*, the latency remains small but grows for 1000 *tuples/s*.

6 Conclusions

We presented an integrated approach for scale out and failure recovery through explicit state management of stateful operators. Our approach treats operator state as an independent entity, which can be checkpointed, backed up, restored and partitioned by the SPS. Based on these operations, the SPS can support dynamic scale out of operators while being fault tolerant.

Our results show that our approach can be used effectively to provision Amazon EC2 resources against increasing input rates in the Linear Road Benchmark and also support

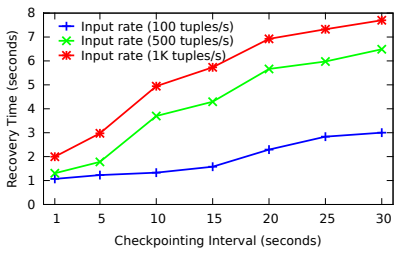


Figure 8 Recovery time for different R+SM checkpointing intervals.

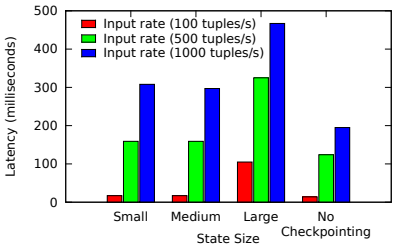


Figure 9 Overhead of state checkpointing for different input rates and state sizes

open loop workloads. Despite the state checkpointing, processing latency remains within desired levels.

As future work, we plan to extend our scale out policy with support for scale in to enable truly elastic deployments of cloud-based SPSs.

Acknowledgements This work was supported by a PhD CASE Award funded by the Engineering and Physical Sciences Research Council (EPSRC) and BAE Systems.

References

1 Daniel J Abadi, Y Ahmand, et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

2 D Agrawal, S Das, et al. Big Data and Cloud Computing: Current State and Future Opportunities. In *EDBT*, 2011.

3 Arvind Arasu, Mitch Cherniack, et al. Linear Road: A Stream Data Management Benchmark. In *VLDB*, 2004.

4 Raul Castro Fernandez, Matteo Migliavacca, et al. Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management. In *SIGMOD*, 2013.

5 Phillipa Gill, Navendu Jain, et al. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *SIGCOMM*, 2011.

6 L Neumeyer, B Robbing, et al. S4: Distributed Stream Computing Platform. In *ICDMW*, 2010.

7 Nish Parikh and Neel Sundaresan. Scalable and Near Real-Time Burst Detection from eCommerce Queries. In *SIGKDD*, 2008.

8 M Russell. *Mining the Social Web*. O'Reilly, 2011.

9 Benjamin Satzger, Waldemar Hummer, et al. Esc: Towards an Elastic Stream Computing Platform for the Cloud. In *IEEE CLOUD*, 2011.

10 Scott Schneider, Henrique Andrade, et al. Elastic Scaling of Data Parallel Operators in Stream Processing. In *IPDPS*, 2009.

11 Zoe Sebepou and Kostas Magoutis. CEC: Continuous Eventual Checkpointing for Data Stream Processing Operators. In *DNS*, 2011.

12 Twitter Storm. `github.com/nathanmarz/storm/wiki`.

13 Rohit Wagle, Henrique Andrade, et al. Distributed Middleware Reliability and Fault Tolerance Support in System S. In *DEBS*, 2011.

14 Erik Zeitler and Tore Risch. Massive Scale-out of Expensive Continuous Queries. *VLDB Endowment*, 4(11), 2011.

15 Zhe Zhang, Yu Gu, et al. A Hybrid Approach to HA in Stream Processing Systems. In *ICDCS*, 2010.

Generalizing Multi-Context Systems for Reactive Stream Reasoning Applications*

Stefan Ellmauthaler

Intelligent Systems, Institute of Computer Science, Leipzig University
P.O. Box 100920, 04009 Leipzig, Germany
ellmauthaler@informatik.uni-leipzig.de

Abstract

In the field of artificial intelligence (AI), the subdomain of knowledge representation (KR) has the aim to represent, integrate, and exchange knowledge in order to do some reasoning about the given information. During the last decades many different KR-languages were proposed for a variety of certain applications with specific needs. The concept of a managed Multi-Context System (mMCS) was introduced to provide adequate formal tools to interchange and integrate knowledge between different KR-approaches. Another arising field of interest in computer science is the design of online applications, which react directly to (possibly infinite) streams of information. This paper presents a genuine approach to generalize mMCS for online applications with continuous streams of information. Our major goal is to find a good tradeoff between expressiveness and computational complexity.

1998 ACM Subject Classification 1.2.11 Distributed Artificial Intelligence

Keywords and phrases Knowledge Representation, Artificial Intelligence

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.19

1 Introduction

Research in the field of knowledge representation has originated a large variety of formats and languages. To use those formal concepts a wealth of tools have emerged (e.g. databases, ontologies, triple-stores, modal logics, temporal logics, nonmonotonic logics, logic programs under nonmonotonic answer set semantics, ...). Those tools were designed for specific needs of certain applications in mind. With the idea of a *"connected world"*, nowadays we do not intend to divide information over different applications. It is desirable to have all information available for every application if need be. To express all of this knowledge, represented in specifically tailored languages, in a universal language would be too hard to achieve from the point of view of complexity as well as the troubles arising from the translation of the representations.

A second issue in current knowledge representation, which is already addressed in different fields of knowledge representation (e.g. stream data processing and querying [10, 9], stream reasoning with answer set programming [6], forgetting in general [8, 5]), is the lack of *online* usage of KR tools and formalisms. Most of the approaches only assume one-shot computations, which is triggered by a user. This may be a specific request in the form of a query to a computer. In practice there are many applications where knowledge is provided in a constant flow of information and it is desired to reason over this knowledge in a continuous manner.

* This research has been funded by DFG (project FOR 1513)

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 19–26

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICCSW

The concept of nonmonotonic Multi-Context Systems (MCS) [2] is a promising approach to achieve a formalism which will not suffer from any of the two shortcomings of current KR-languages. The problem of connecting divided knowledge was the motivation of MCS and its successor [4]. In the following we want to generalize those mMCS to be *reactive* to their environment.

The paper proceeds as follows. After providing some motivating examples for an application of reactive managed Multi-Context Systems in Section 2, we will give an overview on the necessary background regarding managed Multi-Context Systems in Section 3. Section 4 will then introduce the new reactive concepts as an extension to managed Multi-Context Systems. A conclusion with possible future work and a discussion of related work concludes the paper.

2 Motivation

In this section we want to describe one specific application, where a reactive version of MCS would be beneficial. Although our new concept was intended to work for this special use-case, the present approach provides a general and abstract formalism for the whole variety of online-applications.

2.1 Assisted Living

In general we mean by Assisted Living some sort of intelligent apartment, which tries to analyze the behavior of its inhabitants to support them in their daily living. To be more precise, one application of Assisted Living could be the detection of emergencies that may occur in the apartment. As an example imagine a kitchen with different sensors installed. One severe emergency would be that the resident forgot to turn off the cooking stove. Then the intelligent system should react accordingly and either turn it off by itself or by giving an adequate reminder to the resident. Another example could be the detection of an accident. In case one inhabitant had a heart attack or got injured badly, the intelligent apartment should detect it and launch an appropriate emergency-measure (e.g. emergency call). Another convenience-increasing action that could be taken by the apartment is to detect whether the inhabitant wants to be disturbed or not (e.g. he is sleeping). Based on this knowledge it could become handy to mute the mobile phone to avoid an unwanted interruption. But it would be wise to enable the sound again if someone important is calling or the alarm clock wants to awake the inhabitant. These examples are only few possibilities and they may be extended by additional interaction of the apartment (e.g. by giving it access to robots and similar mechanics).

2.2 Realization

The above described apartment may be realized by the installation of different sensors in each room. Some possibilities would be: cameras, microphones, pressure plates, thermostats, and power meters. Each of these sensors will provide a constant flow of information (i.e. a stream). For one apartment an intelligent agent will reason about these streams and conclude on the current behavior of the inhabitant (e.g. if the inhabitant is in the kitchen and the cooking stove is on then he will cook something). Due to the high amount of information given by the stream of each sensor, it is now desirable to get some preprocessing done by the sensors before they send their information to the agent (e.g. the camera detects movement or identifies objects). To get more sophisticated information it is now imaginable to group

a set of sensors to an own agent with its own reasoning (e.g. all sensors in the kitchen, all sensors that track movement, ...). Then it is the task for the apartment-agent to find reasonable conclusions based on the information delivered by the different sensors/agents. Those conclusions can be the current activity of one inhabitant and the appropriate reactions by the agent itself. Due to the possibility of wrong sensor-data, previously drawn conclusions which are refuted, and other inconsistencies/conflicts between the different streams, it is now important to find some kind of equilibria between those agents in a similar way as it is described for Multi-Context Systems [3, 4, 1].

In addition, the agent may encounter many situations with exceptions. One example could be that an inhabitant is cooking and during the waiting time he goes to the restroom. In this case the apartment would not be asked for detecting an emergency. It may also not be an emergency if the inhabitant is going to watch television during the cooking time. But it is an emergency situation if he falls asleep during watching television and will not awake when the meal is done.

3 Background

In this section we will present the already existing definitions for managed Multi-Context Systems (mMCS) [4]. Intuitively, the management extension of MCS changes the bridge rules of the MCS in such a way, that the head of the bridge rule is an arbitrary operator. At first we need to define a logic suite, which allows dynamic changes of the context semantics.

▶ **Definition 1.** A *logic suite* $LS = (\mathcal{BS}_{LS}, \mathcal{KB}_{LS}, \mathcal{ACC}_{LS})$ consists of the set $\mathcal{BS}_{LS}$ of possible belief sets, the set $\mathcal{KB}_{LS}$ of well-formed knowledge-bases, and a nonempty set $\mathcal{ACC}_{LS}$ of possible semantics of LS, i.e. $ACC_{LS} \in \mathcal{ACC}_{LS}$ implies $ACC_{LS} : \mathcal{KB}_{LS} \to 2^{\mathcal{BS}_{LS}}$.

Each logic suite LS has a set of formulas $F_{LS} = \{s \in kb \mid kb \in \mathcal{KB}_{LS}\}$ which represent all formulas occurring in its knowledge base. To describe which operators are allowed, we use a *management base* OP, which is a set of operation names. For each logic suite LS and management base OP, let $F_{LS}^{OP} = \{o(s) \mid o \in OP, s \in F_{LS}\}$ be the set of operational statements which can be built from OP and F_{LS}. The semantics of statements in F_{LS}^{OP} is defined in terms of a *management function*. It allows to modify formulas in a context (e.g. by addition, removal, ...) as well as any desired operation to be applied on a formula or a context.

▶ **Definition 2.** A *management function* over a logic suite LS and a management base OP is a function $mng : 2^{F_{LS}^{OP}} \times \mathcal{KB}_{LS} \to 2^{\mathcal{KB}_{LS} \times \mathcal{ACC}_{LS}} \setminus \{\emptyset\}$.

▶ **Definition 3.** A *managed Multi-Context System* M is a collection $(C_1, \ldots, C_n)$ of managed contexts where, for $1 \leq i \leq n$, each managed context C_i is a quintuple $C_i = (LS_i, kb_i, br_i, OP_i, mng_i)$ such that

- $LS_i = (\mathcal{BS}_{LS_i}, \mathcal{KB}_{LS_i}, \mathcal{ACC}_{LS_i})$ is a logic suite,
- $kb_i \in \mathcal{KB}_{LS_i}$ is a knowledge base,
- OP_i is a management base,
- br_i is a set of bridge rules for C_i, with the form

$$op_i \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), not(c_{j+1} : p_{j+1}), \ldots, not(c_m : p_m).$$

  such that $op_i \in F_{LS_i}^{OP_i}$ and for all $1 \leq k \leq m$ there exists a context $c_k \in (C_1, \ldots, C_n)$ such that $p_k \in S \in \mathcal{BS}_{LS_{c_k}}$, and
- mng_i is a management function over LS_i and OP_i.

For a bridge rule $r \in br_i$ we will use $op(r)$ to denote the operator $op_i \in F_{LS_i}^{OP_i}$ and $body(r)$ denotes the set $\{(c_{k_1} : p_{k_1}) \mid 1 \leq k_1 \leq j\} \cup \{not(c_{k_2} : p_{k_2}) \mid j < k_2 \leq m\}$.

A *belief state* $S = (S_1, \ldots, S_n)$ of M is a belief set for every context, i.e. $S_i \in \mathcal{BS}_{LS_i}$. We denote the set of applicable operations by $app_i(S) = \{op(r) \mid r \in br_i \wedge S \models body(r)\}$. The term of *equilibrium* is used to define the semantics of an mMCS.

▶ **Definition 4.** Let $M = (C_1, \ldots, C_n)$ be an mMCS. A belief state $S = (S_1, \ldots, S_n)$ is an equilibrium of M iff for every $1 \leq i \leq n$ there exists some $(kb_i', ACC_{LS_i}) \in mng_i(app_i(S), kb_i)$ such that $S_i \in ACC_{LS_i}(kb_i')$.

4 Reactive Managed Multi-Context Systems

In the following we will present different approaches to a reactive managed Multi-Context System. At first we will try to get a generalization of the already existing approach of managed Multi-Context Systems. Afterwards we will propose a less complex approach for faster reactions to incoming information. Finally we will combine both variants to gain a solution which benefits from both approaches.

4.1 Preference-Based Iterative Managed Multi-Context System

This part will sketch what needs to be added to the current approach of mMCS to make it suitable for the previously given applications. We have chosen a similar approach to the reactive concept as it was applied by Schaub et al. [6, 7] for their Answer Set Programming-Solver. So we will manipulate our knowledge bases iteratively, based on the current equilibria which may take different input stream information into account. Therefore we will refer to it as an *iterative managed Multi-Context System (imMCS)*. For easier recognition of the different tasks, we will introduce different types of contexts. We will need at least three of them:

- *observing contexts:* these contexts are connected via sensors to the outside world and obtain new information constantly.
- *reasoning contexts:* these contexts are internal modules. It is important that those are not connected to sensors and so they do rely on the information given by other contexts. They are responsible to interpret the different observations and determine what is going on, i.e. are things working properly, does an action need to be taken.
- *control contexts:* this context has the role to do some kind of meta-reasoning for the imMCS. Its role is to:
  1. set sliding windows for other contexts[1],
  2. set inconsistency handling policies (e.g. take sensor reliability into account in case of inconsistencies),
  3. set the used semantics and reasoning modes,
  4. determine necessary actions[2] (e.g. start an alarm), and
  5. decide which contexts need to re-reason (e.g. after a change done by the control context) or which context shall be idle.

To model the dynamic development of equilibria over time, we introduce the notion of a *run of an mMCS*. Intuitively a run describes the provided knowledge and the computed equilibria at a given time.

[1] Reactive reasoning mechanisms use sliding windows to handle possibly infinite streams (c.f. [6]).
[2] It would be reasonable to use such control contexts as the way to communicate with the real world

▶ **Definition 5.** Let M be a managed MCS with contexts $C = (C_1, \ldots, C_n)$ ($C_1, \ldots, C_k$ are observer contexts). Let $Obs = (Obs^0, Obs^1, \ldots)$ be a sequence of observations, that is, for $j \geq 0$, $Obs^j = (Obs_i^j)_{i \leq k}$, where Obs_i^j is the new (sensor) information for context i at step j, which is formalized as sets of formulas. A run R of M induced by Obs is a sequence

$$R = Kb^0, Eq^0, Kb^1, Eq^1, \ldots$$

where

- $Kb^0 = (Kb_i^0)_{i \leq n}$ is the collection of initial knowledge bases, Eq^0 an equilibrium of Kb^0,
- for $j \geq 1$ and $i \leq n$, Kb_i^j is the knowledge base of context C_i produced by the context's management function for the computation of Eq^{j-1}, and $Kb^j = (Kb_i^j)_{i \leq n}$,
- for $j \geq 1$, Eq^j is an equilibrium for the knowledge bases

$$(Kb_0^j \cup Obs_0^j, \ldots, Kb_k^j \cup Obs_k^j, Kb_{k+1}^j, \ldots, Kb_n^j).$$

We call $M' = (C, Obs)$ an *iterative managed MCS* (imMCS).

Note that there may be more than one equilibrium, which would lead to different knowledge bases at the next step of the run. To avoid this multiplication of underlying knowledge, we need to introduce a method to reduce the number of possible equilibria. For this task there are different possible approaches:

1. usage of brave and cautious reasoning methods[3] for the selection of the applicable operations on the contexts.
2. preferences over the bridge rules to get a preferred equilibrium.

In general it may lead to side effects when using brave reasoning for the selection of applicable operations. For example one equilibrium may add a positive literal to a knowledge base, while another equilibrium would add the negated literal. That would lead obviously to an inconsistency although both equilibria were consistent. On the other hand cautious reasoning may lead to a situation where some crucial consistency preserving operations may be missing (c.f. Example 6).

▶ **Example 6.** Let Kb_1^0 be an initial knowledge base in an imMCS M'. The operation *insert* (resp. *revoke*) adds (resp. removes) formulas to (resp. from) the knowledge base. Suppose the negated literals $\neg a$ and $\neg b$ are both in Kb_1^0. The computation of the equilibria results in two sets of belief states $\{Eq_1^0, Eq_2^0\} = Eq^0$, where $app_1(Eq_1^0) = \{revoke(\neg a), insert(a \vee b)\}$ and $app_1(Eq_2^0) = \{revoke(\neg b), insert(a \vee b)\}$. With cautious reasoning only the operation $insert(a \vee b)$ would be executed, which would result in an inconsistent knowledge base.

To ensure that only one equilibrium remains, we will introduce the preference function $pref_i$. Each context provides this function, which takes the set of equilibria and returns a strict total order over them. In addition the whole Multi-Context System provides the preference function *pref*, which takes the total orderings and returns one unique equilibrium.

▶ **Definition 7.** Let $M = (C, Obs)$ be an imMCS and $\mathcal{EQ}$ be a set of equilibria. $M_p = (C, Obs, pref)$ is a preference based imMCS *(pimMCS)* where

- each context C_i has a function $\text{pref}_i : \mathcal{EQ} \to total(\mathcal{EQ})$, where $total(\mathcal{EQ}) = \{R \subseteq \mathcal{EQ} \times \mathcal{EQ} \mid R \text{ is strictly totally ordered}\}$ to associate a strict total ordering of equilibria to each context, and
- the function $pref : (pref_1(\mathcal{EQ}), \ldots, pref_n(\mathcal{EQ})) \mapsto Eq$ returns exactly one equilibrium $Eq \in \mathcal{EQ}$.

[3] Intuitively, given alternative sets of beliefs, for brave reasoning it is sufficient that one belief set supports a conclusion, while cautious reasoning requires that each belief set supports a conclusion

Intuitively, each context propagates its most appreciated equilibria. Afterwards the Multi-Context System determines the "best" fitting equilibrium. Note that at this point we do not intend to give a semantic definition for those two functions. How these equilibria are ordered and how the equilibrium is selected remains adjustable to the specific instance of the Multi-Context System.

4.2 Reactive Bridge Rules

In general the computation of equilibria is expensive [4]. It was shown that the identification of a global equilibria is always one level higher on the polynomial hierarchy than the computation of belief sets of the context with the hardest problem. Due to this potentially high amount of computation time, we present another approach, which will not utilize the concept of global equilibria. The intuitive idea behind our *Reactive Bridge Rules* (RBR) is to provide rules to add supplementary information to the input stream of another reactive context. These rules are evalutated over the belief sets of the different contexts. To control how "informative" one of those rules is, it can be specified for each rule whether its literals need to occur in one or every belief set of the context.

▶ **Definition 8.** A *Reactive Bridge Rule* (RBR) r for a context C_i of a collection of n contexts is a rule of the form

$$t, j : h \leftarrow b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m$$

where

- $t \in \{b, c\}$ specifies whether the literals need to be evaluated bravely or cautiously,
- $j \leq n$ specifies which context will be provided with additional information,
- h is some information which may be added to the input stream of C_j, and
- for $l \leq m$, b_l is a literal.

We will denote the body of one RBR r as *body(r)*, all positive literals b_l, where $l \leq k$ as $b^+(r)$, and all negated literals b_l, where $k < l \leq m$ as $b^-(r)$. Based on the given evaluation mode of the rule, there are different semantics to be applied to the rule. Note that it is obligatory for each context that it has an input stream.

▶ **Definition 9.** Let r be an RBR of a context C_i, $ACC_{LS_i} \in \mathcal{ACC}_{LS_i}$ be a selected semantics, and $S = \{S_1 \ldots, S_j\}$ be the belief sets of C_i at the step t, such that $S = ACC_{LS_i}(kb_i^t)$, where kb_i^t is the knowledge base of context C_i at step t.

- If r is a cautious RBR, it is satisfied if $\forall_{B \in S}(b^+(r) \subseteq B \wedge b^-(r) \cap B = \emptyset)$.
- If r is a brave RBR, it is satisfied if $\exists_{B \in S}(b^+(r) \subseteq B \wedge b^-(r) \cap B = \emptyset)$.

If a rule r is satisfied, then h will be added to the input stream of the context C_j at step $t + 1$.

We will write $\mathcal{RBR}_i^j$ to denote the set of added information to the input stream of context i at step $j+1$, based on the belief sets of step j. Intuitively a RBR wants to inform other contexts of the outcome of different conclusions drawn by a context, based on its observations. The two types of rules were chosen to distinguish between possible conclusions which may be very important and those conclusions which can be drawn safely. In our assisted living scenario there may be events which are more critical than others. For example the possibility of an emergency should be considered as soon as possible, even if it is not assured in every belief set of a context. On the other hand some conclusions may not be neccessary to be forwarded to another context. One example could be the control of the door lock. The door should only open for visitors if every belief set is sure that the person may enter the assisted living environment.

4.3 Combination Of Both Concepts

The two newly introduced concepts have their advantages and disadvantages. The pimMCS do compute equilibria and therefore it is required that all involved contexts agree on a decision. Alas, their computation is quite expensive. Thus it may happen that the computation of an equilibrium takes very long compared to the intervals of newly arriving information in the input streams. With infinite data streams in mind this is a serious issue. The use of sliding windows will force that older, but probably important information is lost due to its size. In case the window is extended automatically to be able to fit all new information since the last equilibria-computation in the run, this memory will grow larger the longer the computation takes. In addition a larger window may also increase the time effort of the next computation of the equilibria, which is some kind of a vicious cycle.

On the other hand RBRs only need the belief sets of each context and there is no need for any agreement on their conclusions and beliefs. This computation involves no communication between the contexts and further it is not neccessary to find an equilibrium. Of course the results are not as strong as an equilibrium, as there is no commonly acceptable belief set of the problem and only local points of view about them.

Now we want to combine both approaches to achieve a *Reactive Managed Multi-Context System (rmMCS)*, which takes the advantages of both ideas and avoids their disadvantages. In general it is desirable to get a formal system which computes equilibria on which decisions are done. Our idea is to compute a run for a pimMCS, where each context has an input stream. During the computation of an equilibrium each context can agree with, RBRs are allowed to manipulate the input streams of the contexts. We also allow each context to change its belief sets based on new stream information, such that another set of RBRs is allowed to manipulate the streams further. Note that this manipulation and change of beliefs shall not affect the computation of the equilibrium. Intuitively, it can be seen as a parallel process.

▶ **Definition 10.** Let M be a preference based iterative managed MCS with contexts $C_0, \ldots, C_n$, context-specific preferences $pref_0, \ldots, pref_n$ and a global preference $pref$. Let $Is = (Is^0, Is^1, \ldots)$ be a sequence of input streams, that is, for $j \geq 0$, $Is^j = (Is_i^j)_{i \leq n}$, where Is_i^j is the current input stream for context i at step j. Let $f(Eq^i)$ be a function that returns the step where the computation of the equilibrium of step i finished, and RBR be a set of reactive bridge rules. A run R of M induced by IS is a sequence

$$R = Kb^0, Eq^0, Kb^{f(Eq^0)}, Eq^{f(Eq^0)}, \ldots$$

where

- $Kb^0 = (Kb_i^0)_{i \leq n}$ is the collection of initial knowledge bases, Eq^0 an equilibrium of Kb^0,
- for $i \leq n$, $mIs_i^0 = Is_i^0$ is the modified initial input stream,
- for $j \geq 1$ and $i \leq n$, $mIs_i^j = Is_i^j \cup \mathcal{RBR}_i^{j-1}$ is the modified input stream at step j,
- for $j \geq 0$, Eq^j is the preferred equilibrium at step j,
- for $j \geq 1$ and $i \leq n$, Kb_i^j is the knowledge base of context C_i produced by the context's management function for the computation of Eq^k, such that $f(Eq^k) = j$, and $Kb^j = (Kb_i^j)_{i \leq n}$, and
- for $j \geq 1$, Eq^j is an equilibrium for the knowledge bases

$$(Kb_0^j \cup mIs_0^j, \ldots, Kb_n^j \cup mIs_n^j).$$

We call $M' = (M, Is, RBR)$ a reactive managed Multi-Context System.

5 Conclusion & Future Work

In this paper we have presented two generalizations for managed Multi-Context Systems, which can utilize streams containing information. We want to mention again that we had stream reasoners, such as `oclingo`[6] as contexts in mind. However, the presented formalism may work well with different approaches. Our goal with this new formalism is to provide a framework where as many formalisms as possible may be used as contexts.

Intended future work is an instantiation of the formalism, to model the given application of assisted living. In addition it will be necessary to investigate possible side effects of the RBR with respect to the rmMCS. In this field there are also some questions which are not answered in this paper (e.g. how should the preference functions be handled). Another interesting field is in general the usage of other formalisms. Are there any undesired effects if we use e.g. C-SPARQL [9]. Is it important to restrict our systems in any way to such that their underlying contexts are not affected in an undesired way. Additionally it is open on how to utilize KR formalisms and tools which do not support online reasoning. Is it sufficient to provide some kind of reactive add-on, which communicates between a stream and a *one-shot offline* formalism?

References

1 Gerhard Brewka. Multi-context systems: Specifying the interaction of knowledge bases declaratively. In Markus Krötzsch and Umberto Straccia, editors, *RR*, volume 7497 of *Lecture Notes in Computer Science*, pages 1–4. Springer, 2012.

2 Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI*, pages 385–390. AAAI Press, 2007.

3 Gerhard Brewka, Thomas Eiter, and Michael Fink. Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 233–258. Springer, 2011.

4 Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl. Managed multi-context systems. In Toby Walsh, editor, *IJCAI*, pages 786–791. IJCAI/AAAI, 2011.

5 Fu-Leung Cheng, Thomas Eiter, Nathan Robinson, Abdul Sattar, and Kewen Wang. Lp-forget: A system of forgetting in answer set programming. In Abdul Sattar and Byeong Ho Kang, editors, *Australian Conference on Artificial Intelligence*, volume 4304 of *Lecture Notes in Computer Science*, pages 1101–1105. Springer, 2006.

6 Martin Gebser, Torsten Grote, Roland Kaminski, Philipp Obermeier, Orkunt Sabuncu, and Torsten Schaub. Stream reasoning with answer set programming: Preliminary report. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *KR*. AAAI Press, 2012.

7 Martin Gebser, Orkunt Sabuncu, and Torsten Schaub. An incremental answer set programming based system for finite model computation. *AI Communications*, 24(2):195–212, 2011.

8 Jérôme Lang and Pierre Marquis. Reasoning under inconsistency: A forgetting-based approach. *Artif. Intell.*, 174(12-13):799–823, 2010.

9 Danh Le-Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth. Linked stream data processing. In Thomas Eiter and Thomas Krennwallner, editors, *Reasoning Web*, volume 7487 of *Lecture Notes in Computer Science*, pages 245–289. Springer, 2012.

10 Carlo Zaniolo. Logical foundations of continuous query languages for data streams. In Pablo Barceló and Reinhard Pichler, editors, *Datalog*, volume 7494 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2012.

Conformal Prediction under Hypergraphical Models*

Valentina Fedorova, Alex Gammerman, Ilia Nouretdinov, and Vladimir Vovk

Computer Learning Research Centre
Royal Holloway, University of London, UK
{valentina,ilia,alex,vovk}@cs.rhul.ac.uk

Abstract

Conformal predictors are usually defined and studied under the exchangeability assumption. However, their definition can be extended to a wide class of statistical models, called online compression models, while retaining their property of automatic validity. This paper is devoted to conformal prediction under hypergraphical models that are more specific than the exchangeability model. We define conformity measures for such hypergraphical models and study the corresponding conformal predictors empirically on benchmark LED data sets. Our experiments show that they are more efficient than conformal predictors that use only the exchangeability assumption.

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases conformal prediction, hypergraphical models, conformity measure

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.27

1 Introduction

The method of conformal prediction was introduced and is usually used for producing valid prediction sets under the exchangeability assumption; the validity of the method means that the probability of making a mistake is equal to (or at least does not exceed) a prespecified significance level ([5], Chapter 2). However, the definition of conformal predictors can be easily extended to a wide class of statistical models, called online compression models (OCMs; [5], Chapter 8). OCMs compress data into a more or less compact summary, which is interpreted as the useful information in the data. With each "conformity measure", which, intuitively, estimates how well a new piece of data fits the summary, one can associate a conformal predictor, which still enjoys the property of automatic validity.

This paper studies conformal prediction under the OCMs known as hypergraphical models ([5], Section 9.2). Such models describe relationships between data features. In the case where every feature is allowed to depend in any way on the rest of the features, the hypergraphical model becomes the exchangeability model. More specific hypergraphical models restrict the dependence in some way. Such restrictions are typical of many real-world problems: for example, different symptoms can be conditionally independent given the disease. A popular approach to such problems is to use Bayesian networks (see, e.g., [2]). The definition of Bayesian networks requires a specification of both the pattern of dependence between features and the distribution of the features. Usual methods guarantee a valid probabilistic outcome if the used distributions of features are correct. Several algorithms (see, e.g., [2],

* A longer version of this paper appeared in the proceedings of COPA **2013** [4].

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 27–34

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Chapter 9) are known for estimating the distribution of features; however, the accuracy of such approximations is a major concern in applying Bayesian networks. The conformal predictors constructed from hypergraphical OCMs use only the pattern of dependence between the features but do not involve their distribution. This makes conformal prediction based on hypergraphical models more robust and realistic than Bayesian networks.

As far as we know, conformal prediction has been studied, apart from the exchangeability model and its variations, only for the Gauss linear model and Markov model (see [5], Chapter 8, and [3]). Hypergraphical OCMs have been used only in the context of Venn rather than conformal prediction (see [5], Chapter 9).

The rest of the paper is organised as follows. Section 2 formally defines hypergraphical OCMs and briefly reviews their basic properties. Section 3 describes the method of conformal prediction in the context of hypergraphical models and introduces a class of conformity measures for hypergraphical OCMs. Section 4 reports the performance of the corresponding conformal predictors on benchmark LED data sets. Section 5 concludes.

2 Background

Consider two measurable spaces $\mathbf{X}$ and $\mathbf{Y}$; elements of $\mathbf{X}$ are called *objects* and elements of $\mathbf{Y}$ are called *labels*. Elements of the Cartesian product $\mathbf{X} \times \mathbf{Y}$ are called *examples*. A *training set* is a sequence of examples $(z_1, \dots, z_l)$, where each example $z_i = (x_i, y_i)$ consists of an object x_i and its label y_i. The general prediction problem considered in this paper is to predict the label for a new object given a training set. We focus on the case where $\mathbf{X}$ and $\mathbf{Y}$ are finite.

2.1 Hypergraphical Structures

In this paper we assume that examples are structured, consisting of variables. Hypergraphical structures describe relationships between the variables. Formally a *hypergraphical structure*[1] consists of three elements $(V, \mathcal{E}, \Xi)$:

1. V is a finite set; its elements are called *variables*.
2. $\mathcal{E}$ is a finite collection of subsets of V whose union covers all variables: $\bigcup_{E \in \mathcal{E}} E = V$. Elements of $\mathcal{E}$ are called *clusters*.
3. Ξ is a function that maps each variable $v \in V$ into a finite set (of the values that v can take).

A *configuration* on a set $E \subseteq V$ (we are usually interested in the case where E is a cluster) is an assignment of values to the variables from E; let $\Xi(E)$ be the set of all configurations on E. A *table*[2] on a set E is an assignment of natural numbers to the configurations on E. The *size* of the table is the sum of values that it assigns to different configurations. A *table set* is a collection of tables on the clusters $\mathcal{E}$, one for each cluster $E \in \mathcal{E}$. The number assigned by a table set σ to a configuration on E is called its *σ-count*.

[1] The name reflects the fact that the components $(V, \mathcal{E})$ form a hypergraph, where a hyperedge $E \in \mathcal{E}$ can connect more than two vertices.

[2] Generally, a table assigns real numbers to configurations. In this paper we only consider *natural tables*, which assign natural numbers to configurations, and omit "natural" for brevity.

2.2 Hypergraphical Online Compression Models

The example space $\mathbf{Z}$ associated with the hypergraphical structure is the set of all configurations on V. One of the variables in V is singled out as the *label variable*, and the configurations on the label variable are denoted $\mathbf{Y}$. All other variables are *object variables*, and the configurations on the object variables are denoted $\mathbf{X}$. Since $\mathbf{Z} = \mathbf{X} \times \mathbf{Y}$, this is a special case of the prediction setting described at the beginning of this section.

An example $z \in \mathbf{Z}$ *agrees* with a configuration on a set $E \subseteq V$ (or the configuration agrees with the example) if the restriction $z|_E$ of z to the variables in E coincides with the configuration. A table set σ *generated* by a sequence of examples $(z_1, \dots, z_n)$ assigns to each configuration on each cluster the number of examples in the sequence that agree with the configuration; the size of each table in σ will be equal to the number of examples in the sequence, and this number is called the *size* of the table set. Different sequences of examples can generate the same table set σ, and we denote $\#\sigma$ the number of different sequences generating σ.

The *hypergraphical online compression model* (HOCM) associated with the hypergraphical structure $(V, \mathcal{E}, \Xi)$ consists of five elements $(\Sigma, \square, \mathbf{Z}, F, B)$, where:

1. The *empty table set* $\square$ is the table set assigning 0 to each configuration.
2. The set Σ is defined by the conditions that $\square \in \Sigma$ and $\Sigma \setminus \{\square\}$ is the set of all table sets σ with $\#\sigma > 0$. The elements $\sigma \in \Sigma$ are called *summaries*.
3. The *forward function* $F(\sigma, z)$, where σ ranges over Σ and z over $\mathbf{Z}$, updates σ by adding 1 to the σ-count of each configuration which agrees with z.
4. The *backward kernel* B maps each $\sigma \in \Sigma \setminus \{\square\}$ to a probability distribution $B(\sigma)$ on $\Sigma \times \mathbf{Z}$ assigning the weight $\#(\sigma \downarrow z)/\#\sigma$ to each pair $(\sigma \downarrow z, z)$, where z is an example such that, for all configurations which agree with z, the corresponding σ-counts are positive, and $\sigma \downarrow z$ is the table set obtained by subtracting 1 from the σ-counts of the configurations that agree with z. Notice that $B(\sigma)$ is indeed a probability distribution, and it is concentrated on the pairs $(\sigma \downarrow z, z)$ such that $F(\sigma \downarrow z, z) = \sigma$.

We will use "hypergraphical models" as a general term for hypergraphical structures and HOCMs when no precision is required. When discussing hypergraphical models we will always assume that the examples $z_1, z_2, \dots$ are produced independently from a probability distribution Q on $\mathbf{Z}$ that has a decomposition

$$Q(\{z\}) = \prod_{E \in \mathcal{E}} f_E(z|_E) \tag{1}$$

for some functions $f_E : \Xi(E) \to [0, 1]$, $E \in \mathcal{E}$, where z is an example and $z|_E$ its restriction to the variables in E.

2.3 Junction Tree Structures

An important type of hypergraphical structures is where clusters can be arranged into a "junction tree". For the corresponding HOCMs we will be able to describe efficient calculations of the backward kernels. If one wants to use the calculations for a structure that cannot be arranged into a junction tree it can be replaced by a more general junction tree structure before defining the HOCM.

Let (U, S) denote an undirected tree with U the set of vertices and S the set of edges. Then (U, S) is a *junction tree* for a hypergraphical structure $(V, \mathcal{E}, \Xi)$ if there exists a bijective mapping C from the set of vertices U of the tree to the set $\mathcal{E}$ of clusters of the hypergraphical structure that has the following property: $C_u \cap C_w \subseteq C_v$ whenever a vertex v lies on the path from a vertex u to a vertex w in the tree (we let C_x stand for $C(x)$).

If $s = \{u, v\} \in S$ is an edge of the junction tree connecting vertices u and v then C_s stands for $C_u \cap C_v$. It is convenient to identify vertices u and edges s of the junction tree with the corresponding clusters C_u and sets C_s, respectively.

If $E_1 \subseteq E_2 \subseteq V$ and f is a table on E_2, the *marginalisation* of f to E_1 is the table f^* on E_1 assigning to each $a \in \Xi(E_1)$ the number $f^*(a) = \sum_b f(b)$, where b ranges over the configurations on E_2 such that $b|_{E_1} = a$. If σ is a summary then for $u \in U$ denote σ_u the table that σ assigns to C_u, and for $s = \{u, v\} \in S$ denote σ_s the marginalisation of σ_u (or σ_v) to C_s. We will use the shorthand $\sigma_u(z)$ for the number assigned to the restriction $z|_{C_u}$ by the table for the vertex u and $\sigma_s(z)$ for the number assigned to $z|_{C_s}$ by the marginal table for the edge s. Consider the HOCM corresponding to the junction tree (U, S). We use the notation $P_\sigma(z)$ for the weight assigned by $B(\sigma)$ to $(\sigma \downarrow z, z)$. It has been proved ([5], Lemma 9.5) that

$$P_\sigma(z) = \frac{\prod_{u \in U} \sigma_u(z)}{n \prod_{s \in S} \sigma_s(z)}, \tag{2}$$

where n is the size of σ. If any of the factors in (2) is zero then the whole ratio is set to zero.

3 Conformal Prediction for HOCM

Consider a training set $(z_1, \ldots, z_l)$ and an HOCM $(\Sigma, \square, \mathbf{Z}, F, B)$. The goal is to predict the label for a new object x.

A *conformity measure* for the HOCM is a measurable function $A : \Sigma \times \mathbf{Z} \to \mathbb{R}$. The function assigns a *conformity score* $A(\sigma, z)$ to an example z w.r. to a summary σ. Intuitively, the score reflects how typical it is to observe z having the summary σ.

For each $y \in \mathbf{Y}$ denote $\sigma^* \in \Sigma$ the table set generated by the sequence $(z_1, \ldots, z_l, (x, y))$ (the dependence of σ^* on y is important although not reflected in our notation). For $z \in \mathbf{Z}$ such that $\sigma^* \downarrow z$ is defined denote the conformity scores as $\alpha_z := A(\sigma^* \downarrow z, z)$ (notice that $\alpha_{(x,y)}$ is always defined). The *p-value* for y, denoted $p^{(y)}$, is defined by

$$p^{(y)} := \sum_{z : \alpha_z < \alpha_{(x,y)}} P_{\sigma^*}(z) + \theta \cdot \sum_{z : \alpha_z = \alpha_{(x,y)}} P_{\sigma^*}(z) \tag{3}$$

(cf. (8.4) in [5]), where $\theta \sim \mathbf{U}[0, 1]$ is a random number from the uniform distribution on $[0, 1]$, $P_{\sigma^*}(z)$ is the backward kernel, as defined above, and the sums involve only those $z \in \mathbf{Z}$ for which α_z is defined. Then for a significance level ϵ the *conformal predictor* Γ based on A outputs the prediction set

$$\Gamma^\epsilon(z_1, \ldots, z_l, x) := \{y \in \mathbf{Y} : p^{(y)} > \epsilon\}.$$

The following section 3.1 defines one class of conformity measures for HOCMs and section 3.2 describes the criteria for the quality of conformal predictions which we use in the paper; for other conformity measures and more criteria see sections 3.1 and 3.2 in [4].

3.1 Conformity Measures for HOCM

Consider a summary σ and an example (x, y). The conditional probability $P_{\sigma^*}(y \mid x)$ of y given x under P_{σ^*} can be computed using (2) as follows

$$P_{\sigma^*}(y \mid x) = \frac{P_{\sigma^*}((x, y))}{\sum_{y' \in \mathbf{Y}} P_{\sigma^*}((x, y'))},$$

where $\sigma^* := F(\sigma, (x, y))$ and $P_{\sigma^*}((x, y))$ is the backward kernel. Define the *predictability* of an object $x \in \mathbf{X}$ as

$$f(x) := \max_{y \in \mathbf{Y}} P_{\sigma^*}(y \mid x), \tag{4}$$

the maximum of conditional probabilities. If the predictability of an object is close to 1 then the object is "easily predictable". Fix a *choice function* $\hat{y} : \mathbf{X} \to \mathbf{Y}$ such that

$$\forall x \in \mathbf{X} : f(x) = P_{\sigma^*}(\hat{y}(x) \mid x).$$

The function maps each object x to one of the labels at which the maximum in (4) is attained. The *signed predictability conformity measure* is defined by

$$A(\sigma, (x, y)) := \begin{cases} f(x) & \text{if } y = \hat{y}(x) \\ -f(x) & \text{otherwise.} \end{cases} \tag{5}$$

3.2 Criteria for the Quality of Conformal Prediction

In this paper we study the performance of conformal predictors in the online prediction protocol (Protocol 1). Reality generates examples (x_n, y_n) from a probability distribution Q satisfying (1) for some hypergraphical structure. Predictor uses a conformal predictor Γ to output the prediction set $\Gamma^\epsilon_n := \Gamma^\epsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n)$ at each significance level ϵ.

Protocol 1 Online prediction protocol

for $n = 1, 2, \ldots$ **do**
  Reality outputs $x_n \in \mathbf{X}$
  Predictor outputs $\Gamma^\epsilon_n \subseteq \mathbf{Y}$ for all $\epsilon \in (0, 1)$
  Reality outputs $y_n \in \mathbf{Y}$
end for

Two important properties of conformal predictors are their validity and efficiency; the first is achieved automatically and the second is enjoyed by different conformal predictors to a different degree. Predictor *makes an error* at step n if y_n is not in Γ^ϵ_n. The validity of conformal predictors means that, for any significance level ϵ, the probability of error $y_n \notin \Gamma^\epsilon_n$ is equal to ϵ. It has been proved that conformal predictors are automatically valid under their models ([5], Theorem 8.1). In this paper we study problems where the hypergraphical model used for computing the p-values is known to be correct; therefore, the predictions will always be valid, and there is no need to test validity experimentally. One possible way to measure efficiency is to count the *number of multiple predictions* Mult^ϵ_n over the first n steps defined by

$$\mathrm{mult}^\epsilon_n := \begin{cases} 1 & \text{if } |\Gamma^\epsilon_n| > 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathrm{Mult}^\epsilon_n := \sum_{i=1}^{n} \mathrm{mult}^\epsilon_i$$

at each significance level $\epsilon \in (0, 1)$ (cf. [5], Chapter 3). In our experiments we will look at the *percentage of multiple predictions* $\mathrm{Mult}^\epsilon_n / n$; we would like it to be close to 0 for small significance levels.

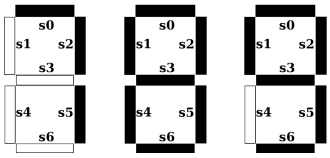


Figure 1 LED images for digits 7, 8, and 9 in the seven-segment display.

4 Experimental Results

4.1 LED Data Set

For our experiments we use benchmark LED data sets generated by a program from the UCI repository [1]. The problem is to predict a digit from an image in the seven-segment display. Figure 1 shows several objects in the data set (these are "ideal images" of digits; there are also digits corrupted by noise). The seven LEDs (light emitting diodes) can be lit in different combinations to represent a digit from 0 to 9. The program generates examples with noise. There is an ideal image for each digit. An example has seven binary attributes $s_0, \ldots, s_6$ (s_i is 1 if the ith LED is lit) and a label c, which is a decimal digit. The program randomly chooses a label (0 to 9 with equal probabilities), inverts each of the attributes of its ideal image with probability $p_{\text{noise}} = 1\%$ independently, and adds the noisy image and the label to the data set.

4.2 Hypergraphical Assumptions for LED Data Sets

We consider two hypergraphical models that agree with the generating mechanism. These models make different assumptions about the pattern of dependence between the attributes and the label; they do not depend on a particular probability of noise p_{noise} or the fact that the same value of p_{noise} is used for all LEDs. For both hypergraphical structures the set of variables is $V := \{s_0, \ldots, s_6, c\}$.

Nontrivial Hypergraphical Model. Consider the hypergraphical structure with the clusters $\mathcal{E} := \{\{s_i, c\} : i = 0, \ldots, 6\}$. A junction tree for this hypergraphical structure can be defined as a chain with vertices $U := \{u_i : i = 0, \ldots, 6\}$ and the bijection $C_{u_i} := \{s_i, c\}$.

Exchangeability Model. The hypergraphical model with no information about the pattern of dependence between the attributes and the label is the exchangeability model. The corresponding hypergraphical structure has one cluster, $\mathcal{E} := \{V\}$. The junction tree is the one vertex associated with V.

4.3 Experiments

For our experiments we create a LED data set with 10000 examples. The data are generated by the program described in section 4.1 with the probability of noise $p_{\text{noise}} = 1\%$.

We consider predictors based on the signed predictability conformity measure (5). The graph with no characters on it corresponds to the idealized predictor and represents an unachievable ideal goal. In the idealized case we know the true distribution for data and use it instead of the backward kernel P_{σ^*} in both (3) and (5). The *pure hypergraphical conformal predictor* (the graph with circles) is obtained using the nontrivial hypergraphical model both when computing p-values (3) and when computing the conformity measure (5). Analogously we use the exchangeability model to obtain the *pure exchangeability conformal predictor* (the

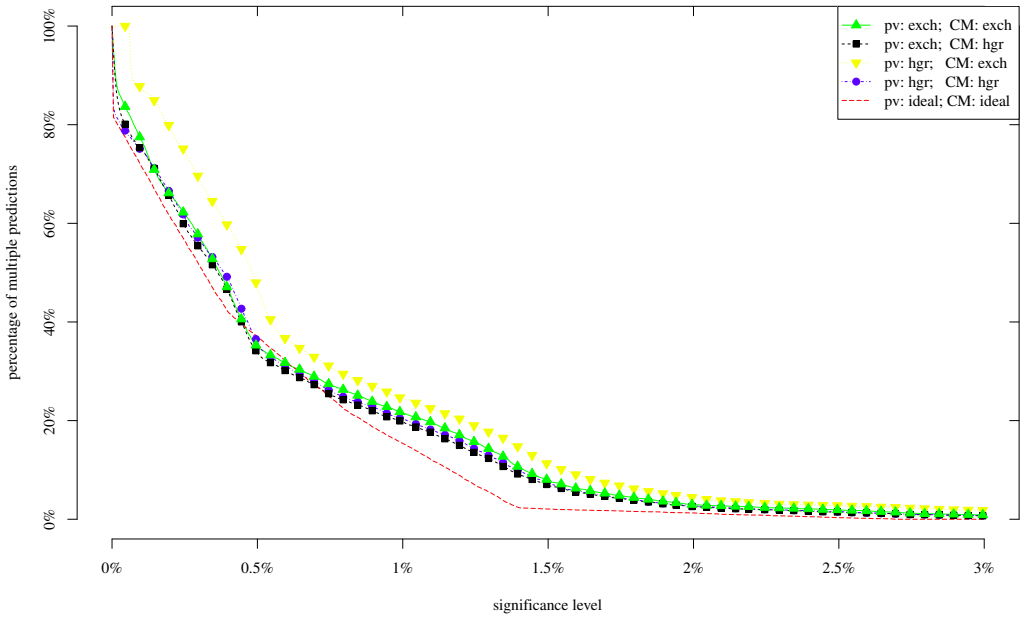


Figure 2 The final percentage of multiple predictions for significance levels between 0% and 3%. The results are for the LED data set with 1% of noise and 10000 examples.

Table 1 The final percentage of multiple predictions in Figure 2 for the significance level 1% and for the graphs with squares and circles.

Seed (10^4)	0	1	...	99	Average	St. dev.
pv: exch; CM: hgr	0.197	0.243	...	0.248	0.192	0.052
pv: hgr; CM: hgr	0.203	0.244	...	0.250	0.196	0.049

graph with triangles point up). The two *mixed conformal predictors* (the graphs with squares and triangles point down) are obtained when we use different models to compute the p-values and the conformity scores. The intuition behind the pure and mixed conformal predictors can be explained using the distinction between hard and soft models made in [6]. The model used when computing the p-values (3) is the hard model; the validity of the conformal predictor depends on it. The model used when computing conformity scores (5) is the soft model; when it is violated, validity is not affected, although efficiency can suffer. The true probability distribution for our generated data conforms to both the exchangeability model and the nontrivial hypergraphical model; so all four conformal predictors are automatically valid, and we study only their efficiency. Figure 2 shows the percentage of multiple predictions $\mathrm{Mult}^{\epsilon}_{10000}/10000$ as function of the significance level $\epsilon \in [0, 0.03]$. In the legend, the hard model used is indicated after "pv" (the way of computing the p-values), and the soft model used is indicated after "CM" (the conformity measure); "exch" refers to the exchangeability model, and "hgr" refers to the nontrivial hypergraphical model. The most interesting graph is the one with squares, corresponding to the exchangeability model as the hard model and the nontrivial hypergraphical model as the soft model. The performance of the corresponding

conformal predictor is typically better than, or at least close to, the performance of any of the remaining realistic predictors. The fact that the validity of the conformal predictor only depends on the exchangeability assumption makes it particularly valuable. The graph with triangles point down corresponds to the nontrivial hypergraphical model as the hard model and the exchangeability model as the soft model; the performance of the corresponding conformal predictor is very poor in our experiments.

Table 1 shows the percentage of the multiple prediction at the significance level 1% for two graphs (with squares and with circles) for several seeds of the pseudorandom number generator. The values of the seed are given in the units of 10,000 (so that 0 stands for 0, 1 for 10,000, 2 for 20,000, etc.). The column "Average" gives the average of all the 100 values, and column "St. dev." gives the standard estimate of the standard deviation computed from those 100 values. The table confirms that the graphs are very close on average (see the penultimate column), but the accuracy of our experiments is insufficient to say which tends to be lower (see the last column).

5 Conclusion

The main finding of this paper is that nontrivial hypergraphical models can be useful for conformal prediction when they are true. More surprisingly, in our experiments they only need to be used as soft models; the performance does not suffer much if the exchangeability model continues to be used as the hard model. This interesting phenomenon deserves a further empirical study.

Acknowledgements We thank the COPA 2013 reviewers for comments that improved the results of the paper. We are indebted to Royal Holloway, University of London, for continued support and funding. This work has also been supported by: the EraSysBio+ grant SHIPREC from the European Union, BBSRC and BMBF; a VLA grant on machine learning algorithms; a grant from the National Natural Science Foundation of China (No. 61128003); a grant from the Cyprus Research Promotion Foundation (research contract TPE/ORIZO/0609(BIE)/24); grant EP/K033344/1 from EPSRC.

References

1 K. Bache and M. Lichman. UCI machine learning repository. School of Information and Computer Sciences, University of California, Irvine, CA, USA, 2013.

2 R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, 1999. Reprinted in 2007.

3 V. Fedorova, I. Nouretdinov, and A. Gammerman. Testing the Gauss linear assumption for on-line predictions. *Progress in Artificial Intelligence*, 1:205–213, 2012.

4 V. Fedorova, I. Nouretdinov, A. Gammerman, and V. Vovk. Conformal prediction under hypergraphical models. In *Proceedings of the Ninth International Conference on Artificial Intelligence Applications and Innovations (AIAI 2013)*, Paphos, Cyprus, 2013. To appear, available at www.alrw.net/articles/09.pdf.

5 V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic Learning in a Random World*. Springer, New York, 2005.

6 V. Vovk, I. Nouretdinov, and A. Gammerman. On-line predictive linear regression. On-line Compression Modelling project (New Series), Working Paper 1, May 2005.

Relational Knowledge Extraction from Attribute-Value Learners

Manoel V. M. França[1], Artur S. D. Garcez[2], and Gerson Zaverucha[3]

1,2 Department of Computing
School of Informatics, City University London
EC1V 0HB London, United Kingdom
manoel.franca.1,aag@city.ac.uk

3 Programa de Engenharia de Sistemas e Computação
COPPE, Universidade Federal do Rio de Janeiro
21941-972 Rio de Janeiro, Brazil
gerson@cos.ufrj.br

Abstract

Bottom Clause Propositionalization (BCP) is a recent propositionalization method which allows fast relational learning. Propositional learners can use BCP to obtain accuracy results comparable with Inductive Logic Programming (ILP) learners. However, differently from ILP learners, what has been learned cannot normally be represented in first-order logic. In this paper, we propose an approach and introduce a novel algorithm for extraction of first-order rules from propositional rule learners, when dealing with data propositionalized with BCP. A theorem then shows that the extracted first-order rules are consistent with their propositional version. The algorithm was evaluated using the rule learner RIPPER, although it can be applied on any propositional rule learner. Initial results show that the accuracies of both RIPPER and the extracted first-order rules can be comparable to those obtained by Aleph (a traditional ILP system), but our approach is considerably faster (obtaining speed-ups of over an order of magnitude), generating a compact rule set with at least the same representation power as standard ILP learners.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving

Keywords and phrases Relational Learning, Propositionalization, Knowledge Extraction

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.35

1 Introduction

Relational learning can be described as the task of learning a first-order logic theory from examples [10, 3]. Inductive Logic Programming (ILP) [15, 17] performs relational learning either directly by manipulating first-order rules or through a method called propositionalization [13, 22], which brings the relational task down to the propositional level by representing subsets of relations as features that can then be used as attributes. In comparison with full ILP, propositionalization normally exchanges accuracy for efficiency [11], as it enables the use of fast attribute-value learners such as rule learners [2], but could lose information in the translation of first-order rules into features. Bottom Clause Propositionalization (BCP) [5] is a recent propositionalization method which allows fast relational learning and also allows propositional learners to obtain accuracy results on par with Inductive Logic Programming (ILP) learners, although differently from ILP learners, what has been learned is not possible to be represented in first-order.

Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 35–42

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we introduce a novel algorithm for consistent extraction of first-order rules from propositional rule learners, when dealing with data propositionalized with BCP. Bottom clauses are variablized first-order clauses that are used as boundaries in ILP hypothesis search space, firstly introduced by Progol [14]. Given an ILP dataset, bottom clauses are built from one positive example e, background knowledge BK (a set of clauses that describe what is known) and language bias L (a set of clauses that define how clauses can be built). A bottom clause is the most specific clause (with most literals) that can be considered a candidate hypothesis. BCP uses bottom clauses for propositionalization because they carry semantic meaning, and because bottom clause literals can be used directly as features in a truth-table, simplifying the feature extraction process [16, 4]. BCP extends Progol's bottom clause generation algorithm to deal with negative examples and it keeps track of a hash table which is responsible to map each constant found during bottom clause generation to an unique variable, for each example. This hash table is one of the key differences between BCP and other propositionalization methods such as RSD [22], SINUS [10] and RELAGGS [12], and is used by our approach to transform propositional rules learned from data which was propositionalized with BCP (which we will refer in this work simply as BCP-rules) to consistent and accurate relational (first-order) rules. A theorem then shows that the extracted first-order rules are consistent with their propositional version.

Our methodology is evaluated using BCP for propositionalization and the propositional rule learner RIPPER on four Alzheimer datasets [9] and we have used three metrics to present our results: standard classification accuracy (the percentage of correctly classified examples, among all tested examples), runtime (total time taken to complete the experiment, from training to testing), and rule size (we define rule size as the number of body and head literals in the entire induced theory). Results show that although information loss is expected when dealing with propositionalization methods [22, 10, 5], the accuracy of both RIPPER and the first-order rules can be comparable with Aleph [20], a traditional ILP system, in some cases, while being considerably faster and generating rules with lower size. The rules generated by our algorithm also have more representational power, by being able to represent disjunctions and negation as failure.

The remainder of this paper is as follows: in Section 2, we review BCP and the propositional rule learner RIPPER. In Section 3, we introduce our contribution: an algorithm for extracting relational rules from BCP-rules. Our empirical results with regard to classification accuracy, runtime and rule size in comparison with RIPPER and Aleph is shown in Section 4, and in Section 5, we conclude and discuss directions for future work.

2 Background

In this section, the key methods and algorithms used in our work are introduced: Bottom Clause Propositionalization and RIPPER.

Bottom Clause Propositionalization [5] is a logic-based propositionalization method which takes advantage of Progol's bottom clause generation algorithm and the previous work from [4], which shows that bottom clause literals can be used as propositional features. In [5], BCP managed to achieve comparable results with Aleph in a number of ILP datasets, even though propositionalization methods incur information loss. Additionally, BCP was faster and also obtained better accuracy and runtime results when compared to RSD. One problem with the obtained results, though, was that all tests were done in the propositional level. We investigate in this paper how to bring those results back to first-order.

BCP has two steps: bottom clause generation and attribute-value mapping. In the first

step, each example is given to Progol's bottom clause generation algorithm [21] to create a corresponding bottom clause representation. To do so, a slight modification is needed to allow the same *hash* function to be shared among all examples, in order to keep consistency between variable associations (i.e., to ensure that variable associations are done in the same way, for different bottom clauses), and to allow negative examples to have bottom clauses as well; the original algorithm deals with positive examples only. The generation algorithm has a single parameter, *depth*, which is the *variable depth* of the bottom clause generation process. A more detailed description of the modified version of Progol's bottom clause generation algorithm can be found in [5].

To illustrate BCP's bottom clause generation, consider the well-known family relationship [15] ILP example: BK = {*mother(mom1, daughter1)*, *wife(daughter1, husband1)*, *wife(daughter2, husband2)*}, positive example *motherInLaw(mom1, husband1)*, and negative example *motherInLaw(daughter1, husband2)*. If the modified bottom clause generation algorithm is executed with $depth = 1$ on the positive and negative examples shown above, it generates the following training set: $S_\perp = \{motherInLaw(A, B)$:- $mother(A, C), wife(C, B)$; $\sim motherInLaw(A, B)$:- $wife(A, C)\}$.

After the creation of the $S_\perp$ set, each bottom clause inside $S_\perp$ is converted into a binary vector v_i, $0 \leq i \leq n$, according to the presence or absence of each found literal inside $S_\perp$, where n is the number of distinct literals inside $S_\perp$.

RIPPER [2] is a well-known propositional rule learner which can be considered the propositional version of the FOIL first-order theory induction algorithm [8] in the sense that it also performs a covering-based algorithm to choose literals to build its theory, using information gain as search heuristic. RIPPER's focus is to tackle noisy data and achieve competitive results with regard to Quinlan's propositional tree-learner C4.5 [18]. RIPPER extends its predecessor IREP [6], by improving its information gain heuristic and its stopping criteria (this improvement was named IREP*), and uses IREP* multiple times, to perform different parts of the learning task. Those parts are: IREP* is used once to obtain an initial rule set, covering part of the positive examples; The rules are optimized with regard to redundancy/consistency; and IREP* is used again to cover the remaining positive examples.

RIPPER has been shown in [2] to generate rules with better performance than C4.5's decision trees and to be efficient (fast and accurate) on large and noisy datasets. RIPPER's ability to generate good rules when dealing with large and noisy datasets is the reason it is chosen to process data which was propositionalized with BCP: bottom clauses can be considerably large, possibly having infinite size [14], and a learner which can deal with large number of noisy features is better suited to deal with BCP.

3 Extracting Relational Knowledge from BCP-Rules

In this section, our algorithm for generating first-order rules from BCP-rules is introduced. It is important to notice that the described methodology below does not require a specific attribute-value learner: any propositional learner which is able to generate logic rules to describe what has been learned, e.g. decision tree learners, rule learners and graph learners, are all able to be used for the proposed rule extraction.

As a first step of our approach, we explain hereafter how BCP-rules are generated. Firstly, BCP is applied in the examples set, generating a bottom clause set $S_\perp$, as shown in Section 2. Then, attribute-value learning takes place. In this work, we have chosen RIPPER due to the advantages described in Section 2, but any learner that can generate rules can be used, although further investigation is required.

By using RIPPER on data propositionalized with BCP, a set of rules is created. From an ILP point of view, those rules do not necessarily obey variable chaining properties or any kind of language bias restrictions: each feature (i.e. each distinct bottom clause atom) is seen as propositional features by RIPPER and thus, further processing needs to be done in order to treat it as first-order. As an example, consider the following propositionalized dataset:

$$\begin{aligned} S_\perp = \{ motherInLaw(A,B) &:- mother(C,B), wife(C,D); \\ motherInLaw(A,B) &:- mother(A,C), wife(C,B); \\ \sim motherInLaw(A,B) &:- wife(C,B), parents(C,B,D), dad(E,F)\}. \end{aligned} \tag{1}$$

From this dataset, one possible rule generated by RIPPER (containing features from positive examples and negated features from negative examples), in Prolog format, could be:

$$R_\perp = \{motherInLaw(A,B) :- mother(A,C), wife(C,D), not(dad(E,F))\}. \tag{2}$$

The first point worth noticing regarding $R_\perp$ is that it can represent the *absence* of a BCP feature, e.g. $not(dad(E,F))$, which is equivalent to negation as failure [7] and shows that the rules we generate have more representational power. The second point is that there is a problem with the generated BCP-rule $R_\perp$, if it is treated directly as first-order: the variables of $dad(E,F)$ are not present in any other body or head atom (from now on, we will refer to those variables as *unconstrained variables*). This is possible to happen due to the fact that all atoms are seen as features generated by BCP, thus not taking into consideration the language bias. If $R_\perp$ is used as theory to infer unseen first-order data, as long as there is at least one $dad/2$ ground atom[1] inside the background knowledge, $dad(E,F)$ would always be true and thus, $not(dad(E,F))$ would always be false and $R_\perp$ would always be false as well, thus limiting the generalization capabilities of $R_\perp$. In order to solve this issue, after generating BCP-rules using RIPPER and obtaining a set of BCP-rules $R_\perp$, all unconstrained variables need to be removed from $R_\perp$ before treating it as a first-order theory.

The process of extracting first-order rules from BCP-rules can be divided into three steps: *unconstrained variables search*, *unconstrained variables replacing* and *first-order filtering*. In the first step, *unconstrained variables search*, a search is done in the BCP-rules to find literals with unconstrained variables (i.e., finding all occurrences of literals such as $not(dad(E,F))$ in $R_\perp$ above). We detect unconstrained variables from the rightmost literal to the leftmost one, by verifying if the variable being checked, belonging to a body literal l_i, appears on any other body literal in $\{l_j | j < i\}$. For each BCP-rule $r \in R_\perp$, we store unconstrained variables (and the literals where they were found) to be used in the next step of our algorithm, *unconstrained variables grounding*.

As an example, let us use the $R_\perp$ defined in (2) as input for *unconstrained variables search*. Firstly, unconstrained variables are searched in the single clause of $R_\perp$, from the right to the left. In the rightmost literal, $not(dad(E,F))$, two unconstrained variables are found: E and F. Because of that, both variables and the literal where they were found are stored. After advancing to the next rightmost literal, one more unconstrained variable is found: D. Thus, D is stored for the next part of our algorithm, together with the literal $wife(C,D)$ where it was found. Note that the other variable, C, is not included, since it appears in $mother(A,C)$ and thus, it is not unconstrained. Since no more unconstrained variables can be found, the first step of our extraction algorithm comes to an end with the following variables/literals stored: $M = \{E, F, not(dad(E,F)); D, wife(C,D)\}$.

[1] Ground atoms are atoms which does not contain variables, only constants.

After that, *unconstrained variables replacing* comes into place. All the stored variables are replaced using the *hash* table generated during BCP, as mentioned on Section 2, in order to eliminate all unconstrained variables which were found in the previous steps. To illustrate that, let us continue our family relationship example. Three variables have been flagged as unconstrained: E and F, from the literal $not(dad(E, F))$, and D, from the literal $wife(C, D)$. Assume the same propositionalized examples set $S_\perp$ shown in (1), that generated the BCP-rules set $R_\perp$. Since the body of the first example of $S_\perp$ contains one variable of M (which is D), the second example does not contain any of the variables mapped in M, and the third example contains all three variables inside M, which are D, E and F (totalizing two occurrences of D and one occurrence of both E and F on $S_\perp$), BCP's *hash* must have two entries for D, one entry for E and one entry for F. Let us assume that those entries are $\{D/husband1\}$, $\{D/daughter1\}$, $\{E/mom1\}$ and $\{F/daughter1\}$.

As explained earlier, as long as a BCP-propositionalized example contains a literal lexically identical to a literal from another example, both will be considered to have the feature represented by that literal. In the case of the feature $wife(C, D)$, for instance, even though different examples have different *hash* mappings for C and D, the presence or not of $wife(C, D)$ is what is considered for propositional learning. This suggests that the rule that defines the truth-value of a feature is a *disjunction over all observed mappings (unifications) in the training set during BCP propositionalization.* Theorem 2 below shows that for each BCP feature, a disjunction over all possible unifications of a feature with grounding operators over all examples is semantically equivalent to the feature itself.

▶ **Definition 1.** Let $e \in E$ be an example of an dataset E, propositionalized with BCP. Also, let f be a BCP feature, let U be the set of all unconstrained variables which can be found inside f, having size k, and let $hash_e$ be the variable/constant mapping generated for example e during BCP. The *grounding unifier* θ_e^f for a feature f with regard to an example e is defined as $\theta_e^f = \{v^1/c^1, v^2/c^2, \cdots, v^k/c^k\}$, where $v_i \in U, 1 \leq i \leq k$ is an unconstrained variable and c_i is the constant which is mapped to v_i, according to $hash_e$. If $k = 0$, $\theta_e^f = \emptyset$.

▶ **Theorem 2.** *Let E be an example set, propositionalized with BCP and having size n, and f be one of the generated features with BCP when applied to E. Also, let $v(f)$ be a valuation function, which associates a boolean truth-value for a feature f. Then, $v(f) \equiv v(f\theta_{e_1}^f) \vee v(f\theta_{e_2}^f) \vee \cdots \vee v(f\theta_{e_n}^f)$, where $\{e_1, e_1 \cdots e_1\} \subset E$ and $f\theta_{e_i}^f$ is the unification of feature f with a grounding unifier $\theta_{e_i}^f$, $1 \leq i \leq n$.*

Proof. Proof by contradiction. Suppose that there exists a feature f and examples $e_i \in E$, $1 \leq i \leq n$, where $v(f) \not\equiv v(f\theta_{e_1}^f) \vee v(f\theta_{e_2}^f) \vee \cdots \vee v(f\theta_{e_n}^f)$ holds. There are two case scenarios that makes this equation true:

- There exists a feature f with truth-value *true*, but all possible unifications of f with ground unifiers $\theta_{e_i}^f, 1 \leq i \leq n$, are false. If f appears in a rule, it must have been found in at least one bottom clause generated with BCP from an example $e \in E$. Then, there exists one set of unifications $\{v/c\}$ in $hash_e$, one for each v inside f, which makes $f\{v/c\}$ true. If this unifier is used as θ_e^f, then at least one member of the disjunction is true.
- There exists a feature f which is false, but at least one possible unification $\theta_{e_i}^f$ of f, $1 \leq i \leq n$, with ground unifier $\theta_{e_i}^f$, is true. Definition 1 ensures that f can be found inside e_i, otherwise $\theta_{e_i}^f$ would not exist. Thus, if $\theta_{e_i}^f$ is a valid unifier for e_i, it also needs to be a valid unifier for f.

◀

We now can solve the problem of BCP-rules having unconstrained variables by replacing them with disjunctions of grounding unifications (we call those unified BCP-rules *constrained BCP-rules*). We illustrate the second step of our algorithm by continuing our family relationship example. From the first step of our relational knowledge extraction algorithm, we have obtained a list of variables that are unconstrained and need to be replaced: E and F, from feature $not(dad(E, F))$, and D, from feature $wife(C, D)$. From (1), one example contains the feature $not(dad(E, F))$ and two contain feature $wife(C, D)$. Thus, $not(dad(E, F))$ is replaced by one grounded literal and $wife(C, D)$ is replaced by a disjunction of two grounded literals, by applying Theorem 2 and using the previously specified *hash* entries for those examples: $\{D/husband1\}$, $\{D/daughter1\}$, $\{E/husband2\}$ and $\{F/daughter1\}$. Those replacements are $\{not(dad(E, F)) \mapsto not(dad(husband1, daughter1))\}$ and $\{wife(C, D) \mapsto wife(C, daughter1) \vee wife(C, husband2)\}$ and thus, the resulting constrained BCP-rule set $R_{\perp}^{C}$ after replacing $R_{\perp}$ from (2) with the created grounded atoms (in prolog format) is

$$R_{\perp}^{C} = \{motherInLaw(A, B) :- mother(A, C), (wife(C, daughter1); wife(C, husband2)), not(dad(husband1, daughter1))\}.$$

Lastly, in *first-order filtering*, we apply a modified version of the theory filtering algorithm T-reduce [20], a companion program to Aleph, in theory $R_{\perp}^{C}$. The original T-reduce algorithm is capable of removing rules that do not cover any first-order training example and rules that contribute negatively to the theory accuracy. We modified T-reduce to also to cut out redundant literals and literals that do not have variables on it. Literals without variables need to be removed for the same reason the unconstrained variables of $not(dad(E, F))$ need to be replaced: depending on the background knowledge, those literals are always true or always false, thus contributing negatively to the rule's ability to generalize. As an example, if our version of T-reduce is applied on $R_{\perp}^{C}$, assuming that $R_{\perp}^{C}$ is non-redundant (otherwise it would be removed by T-reduce), we obtain the final first-order theory $R_{\perp}^{FOL} = \{motherInLaw(A, B) :- mother(A, C), (wife(C, daughter1); wife(C, husband2))\}$, since $not(dad(husband1, daughter1))$ does not have variables on it.

To illustrate the whole process of extracting first-order rules from BCP-rules, our complete procedure is summarized in Algorithm 1. It receives as input a set of BCP-rules $R_{\perp}$ and outputs a set $R_{\perp}^{FOL}$ of extracted first-order rules.

Algorithm 1 First-order Rules Extraction from BCP-rules

1: $R_{\perp}^{FOL} = \emptyset$
2: Let U be the set of unconstrained variables and their respective literals inside $R_{\perp}$
3: **for** each rule r of $R_{\perp}$ **do**
4: Apply Theorem 2 by using U on r to obtain a constrained clause c_r
5: Apply (modified) T-reduce on c_r to obtain a filtered clause $r_{treduce}$
6: Check if $r_{treduce}$ contributes positively towards accuracy; if not, discard it
7: Add $r_{treduce}$ to $R_{\perp}^{FOL}$, if it has not been discarded
8: **end for**
9: **return** $R_{\perp}^{FOL}$

4 Initial Results

In this section, we present the experimental methodology and initial results for our relational knowledge extraction algorithm. We show comparative results between the ILP system

Aleph, RIPPER when trained with BCP-data (we will refer to it as BCP+RIP$_{prop}$) and the extracted first-order rules from BCP+RIP$_{prop}$ (we will refer to it as BCP+RIP$_{FOL}$), using the methodology presented on Section 3. We have used the *Alzheimers* benchmark [9], which consists of four datasets: *Amine*, *Acetyl*, *Memory* and *Toxic*. The used experimental configurations on Aleph, RIPPER and BCP can be found on `http://soi.city.ac.uk/~abdz937/iccsw13Parameters.txt`.

We evaluate the results on three aspects: *standard accuracy*, *runtime* and *theory size*. We define standard accuracy as the percentage of correctly classified examples over test data; we define runtime as the total pre-processing, training and testing times for each system; and we define theory size as the total number of literals (body literals and head literals) in the learned theory. Our obtained results, averaged by 10-fold cross-validation for accuracy and theory size, and accumulated over all 10 fold with regard to runtime, are presented on Table 1. In the accuracy results, values in bold are the highest ones obtained between BCP+RIP$_{prop}$ and BCP+RIP$_{FOL}$ and the difference between them and the ones marked with asterisk (*) are statistically significant by two-tailed, paired t-test.

Table 1 Accuracy results (with standard deviation), runtimes and theory size measurements for the Alzheimers benchmark (accuracies in the first line; runtimes, theory size in the second line). The results accuracy difference between BCP+RIP$_{prop}$ and Aleph, on all four datasets, are not statistically significant. Between BCP+RIP$_{prop}$ and BCP+RIP$_{FOL}$, BCP+RIP$_{FOL}$ managed to obtain statistically comparable accuracy results with BCP+RIP$_{prop}$ in the first two datasets (*Alz-ami* and *Alz-ace*). Comparing directly BCP+RIP$_{FOL}$ with Aleph, BCP+RIP$_{FOL}$ also managed to obtain statistically comparable results on two datasets, *Alz-ami* and *Alz-ace*. Additionally, it can be seen that the our methodology is considerably faster than Aleph, obtaining an average speed-up over all four datasets of more than one order of magnitude, while generating smaller rules as well.

	Alz-ami	***Alz-ace***	***Alz-mem***	***Alz-tox***
Aleph (baseline)	78.71(±5.25) 1:31:05, 36.1	69.46(±4.6) 8:06:06, 47.3	68.57(±5.7) 3:47:55, 45.7	80.5(±4.83) 6:02:05, 37.9
BCP+RIP$_{prop}$	73.35(±4.32) 0:19:49/30	**67.8**(±3.77) 0:23:21/20.1	**65.27**(±7.11) 0:25:11/14.4	**78.44**(±5.44) 0:17:41/35.2
BCP+RIP$_{FOL}$	**77.73**(±4.57) 0:21:59/30.4	63.56(±5.06) 0:26:39/18.7	57.64 ∗ (±5.7) 0:28:45/13.8	66.45 ∗ (±6.93) 0:20:57/18

5 Conclusion and Future Work

This paper has tackled the problem of accurately and consistently represent what has been learned with data previously propositionalized with BCP, but back to a relational format, by introducing a novel algorithm for consistent extraction of first-order rules from propositional rules described with BCP features. A theorem shows that the extracted first-order rules are consistent with their propositional version. Our results show that the presented methodology is promising and it can not only extract accurate rules from propositional learners which used BCP as propositionalization method, but it can also improve its accuracy in the process. Additionally, our approach is capable of generating first-order rules with disjunctions and negation as failure, which standard ILP inducers cannot, and our results show that our approach is considerably faster than Aleph (a speed-up of over an order of magnitude on average) and also generates smaller rules.

As future work, experiments on datasets with numerical (continuous) data, such as Carcinogenesis [19] and with very large data, such as CORA [1] are underway. Also, a new version of RIPPER which satisfies mode declarations, for learning with BCP, is being studied.

References

1 M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. ACM SIGKDD*, pages 39–48, New York, NY, USA, 2003.

2 W. W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart J. Russell, ed., *ICML*, pg. 115–123. Morgan Kaufmann, 1995.

3 L. De Raedt. *Logical and Relational Learning.* Cognitive Technologies. Springer, 2008.

4 F. DiMaio and J. W. Shavlik. Learning an Approximation to Inductive Logic Programming Clause Evaluation. In *ILP*, vol. 3194 of *LNAI*, pg. 80–97, 2004.

5 M. V. M. França, G. Zaverucha, and A. S. D. Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Mach. Learn.*, pg. 1–24, 2013.

6 J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In William W. Cohen and Haym Hirsh, ed., *ICML*, pg. 70–77. Morgan Kaufmann, 1994.

7 M. L. Ginsberg, editor. *Readings in nonmonotonic reasoning.* Morgan Kaufmann, San Francisco, CA, USA, 1987.

8 J. R. Quinlan and R. M. Cameron-Jones. FOIL: A Midterm Report. In *Proc. ECML*, pages 3–20. Springer, 1993.

9 R.D. King and A. Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing*, 13(3-4):411–434, 1995.

10 S. Kramer, N. Lavrač, and P. Flach. Relational Data Mining. chapter Propositionalization approaches to relational data mining, pg. 262–286. Springer, New York, NY, USA, 2000.

11 M. A. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel. Comparative Evaluation Of Approaches To Propositionalization. In *ILP*, vol. 2835 of *LNAI*, pg. 194–217. Springer, 2003.

12 M. A. Krogel and S. Wrobel. Facets of Aggregation Approaches to Propositionalization. pg. 30–39. Department of Informatics, University of Szeged, September 2003.

13 N. Lavrač and S. Džeroski. *Inductive logic programming: techniques and applications.* Ellis Horwood, 1994.

14 S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13(3-4):245–286, 1995.

15 S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

16 S. Muggleton and A. Tamaddoni-Nezhad. QG/GA: a stochastic search for Progol. *Mach. Learn.*, 70:121–133, 2008.

17 S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, vol. 1228 of *LNAI.* Springer, 1997.

18 J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Francisco, CA, USA, 1993.

19 A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. E. Sternberg. Carcinogenesis Predictions using ILP. *Proc. International Workshop on Inductive Logic Programming*, 1297(1297):273–287, 1997.

20 A. Srinivasan. The Aleph System, version 5. http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html, June 2007. Last accessed on July/2013.

21 A. Tamaddoni-Nezhad and S. Muggleton. The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. *Mach. Learn.*, 76(1):37–72, 2009.

22 F. Železný and N. Lavrač. Propositionalization-based Relational Subgroup Discovery With RSD. *Machine Learning*, 62:33–63, 2006.

Tools for the implementation of argumentation models

Bas van Gijzel

Functional Programming Laboratory,
School of Computer Science,
University of Nottingham,
United Kingdom
bmv@cs.nott.ac.uk

Abstract

The structured approach to argumentation has seen a surge of models, introducing a multitude of ways to deal with the formalisation of arguments. However, while the development of the mathematical models have flourished, the actual implementations and development of methods for implementation of these models have been lagging behind. This paper attempts to alleviate this problem by providing methods that simplify implementation, i.e. we demonstrate how the functional programming language Haskell can naturally express mathematical definitions and sketch how a theorem prover can verify this implementation. Furthermore, we provide methods to streamline the documenting of code, showing how literate programming allows the implementer to write formal definition, implementation and documentation in one file. All code has been made publicly available and reusable.

1998 ACM Subject Classification I.2.3 Nonmonotonic reasoning and belief revision

Keywords and phrases argumentation, implementation, functional programming, Haskell, Carneades

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.43

1 Introduction

Argumentation theory is an interdisciplinary field studying how conclusions can be reached through logical reasoning in settings where the soundness of arguments might be subjective and arguments can be contradictory. There are two main approaches: the structured approach giving a predetermined structure to arguments, including for example legal and scientific arguments, while the abstract approach makes no specific assumptions about the form of arguments and is thus generally applicable. Structured argumentation models have seen a recent surge, with new developments in both general frameworks [17, 1, 3] and more domain specific approaches [12, 11]. For the abstract approach, a significant effort has been directed towards the construction of usable tools and efficient implementations; see [4] for a survey. In addition there has been a recent development of translations between structured and abstract argumentation models, allowing an implementer to sidestep the implementation of the structured model by implementing the translation instead and relying on an existing efficient implementation of the translation target. However, despite these tools and existing translations of structured argumentation models into abstract argumentation frameworks in the literature [17, 10, 9, 2], there is a lack of implementations of the structured models.

We give a number of potential reasons:

- Most implementations of structured argumentation models are not publicly available. Simari [18] gives an overview of some of the structured argumentation models, but most

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 43–48

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

implementations are now unavailable or closed source. In the case the code has never been published it means that the information regarding the techniques of implementation are effectively lost, forcing new implementers to develop methods from scratch.
- Translations can be notoriously complex, both in implementation and in verification. As a good example, consider the translation of Carneades to ASPIC$^+$ [10, 9], or the translation of abstract dialectical frameworks to Dung [2]. Both proofs are at least a page long, and are hard to verify even by experts in the field.

To tackle this problem, we introduce a set of methods and tools in Section 2. Then in Section 3 we provide an introduction to the definitions of Carneades each time providing corresponding implementations using previously discussed methods and tools. We conclude in Section 4 with a discussion of our study and how to go from this implementation to an automatic verification in a theorem prover.

2 Tools and methods

2.1 Functional programming

When looking at recent developments in abstract argumentation [4], we can see that answer set programming (ASP) and Prolog have taken a significant role in the efficient implementation and development of general tools. Part of this success can be explained by the paradigm of ASP and logic programming which can express computational problems for Dung's argumentation frameworks [6] in a very natural way, making it possible to make the code partly self-documenting. See also [6], which relates abstract argumentation to logic programming with negation as failure.

For structured argumentation, there has not been such a convincing implementation language yet. There are various implementations done in Java [18], but they are quite far removed from the logical specification making it significantly harder to verify whether the implementation is actually correct. In this paper we instead apply functional programming, using the programming language Haskell [16]. The declarative nature of functional programming, similar to logic programming, is a natural candidate to express structured argumentation frameworks in such a way that the code is close to the actual mathematical definitions [13], while additionally simplifying future verification of such implementations.

2.2 Literate programming

Although implementations of structured argumentation models often have appropriate user instructions, it is less common that such an implementation also documents its methods of implementation. To make this process more attractive, we employ literate programming [14], a technique that allows the user to write both the implementation and documentation, including the formal definitions, in one file. Literate Haskell is Haskell's native version of literate programming, which allows programmers to intermix the writing of LaTeX and Haskell code, while still being readable by a standard Haskell compiler. Additionally, the tool called lhs2tex [15] provides the user with the automatic typesetting of Haskell code within a Literate Haskell file, generating appropriate LaTeX code. This ensures that the documentation is kept up to date along with the programming code.

2.3 Open source and public repositories

As we discussed in Section 1, most implementations of structured argumentation models are not publicly available (anymore) or are closed source. We believe that to progress the know-

ledge of implementing techniques for (structured) argumentation models, implementations should be made publicly available. The implementation in this paper has also been made available through the standard Haskell package repository called Hackage[1], providing source files and automatic generation of html documentation of the API. The public availability and documentation of the implementation has attracted other people to contribute as well, e.g. see Stefan Sabev's github[2] where he extended our implementation for a university module.

2.4 Formalisation in a theorem prover

Given the complexity of some of the structured models and translation we might want to be able to verify the correctness of our implementation. One way to achieve this beyond the proofs done on paper, is to formalise the implementation through an interactive theorem prover of our choice. Haskell, allows for code very close to the mathematics and additionally is of the same functional nature as most theorem provers, making the step from a Haskell program to a theorem prover very natural. We do not have the space to demonstrate this approach here, however an implementation of Dung's argumentation frameworks has been made available online[3]. See [8] for an expanded exposition.

3 A documented implementation of Carneades

In this section we will give definitions of Carneades [12, 11], an argumentation model designed to capture standards and burdens of proof. We discuss the version as given in Gordon and Walton [12], after each definition showing our corresponding implementations in Haskell[4]. Due to space constraints, we do not give the complete implementation, however the source code of this section, is available as a literate programming source file, containing all the left out definitions, corresponding implementations and explanations[5]. This literate programming file can immediately be loaded into the Haskell compiler and is also available as an open source library on Hackage[6]. Although Carneades is very suitable to demonstrate the implementation techniques explained in this paper; it does already have a quite mature implementation available[7].

3.1 Arguments

We strive for a realisation in Haskell that mirrors the mathematical model of Carneades argumentation framework as closely as possible. Ideally, there would be little more to a realisation than a transliteration. In Carneades all logical formulae are literals in propositional logic; i.e., all propositions are either positive or negative atoms. Taking atoms to be strings suffice in the following, and propositional literals can then be formed by pairing this atom with a Boolean to denote whether it is negated or not:

$$\textbf{type } \mathit{PropLiteral} = (\mathit{Bool}, \mathit{String})$$

We write $\overline{p}$ for the negation of a literal p. The realisation is immediate:

[1] `http://hackage.haskell.org/`
[2] `https://github.com/SSabev/Haskell_Carneades`
[3] `http://www.cs.nott.ac.uk/~bmv/Code/AF2.agda`
[4] This section is largely based on previous work in [7].
[5] See `http://www.cs.nott.ac.uk/~bmv/CarneadesDSL/` for the source code and instructions.
[6] `http://hackage.haskell.org/package/CarneadesDSL`
[7] `http://carneades.github.com/`

$negate :: PropLiteral \to PropLiteral$
$negate\ (b, x) = (\neg\ b, x)$

An argument is a tuple of two lists of propositions, its *premises* and its *exceptions*, and a proposition that denotes the *conclusion*:

newtype $Argument = Arg\ ([PropLiteral], [PropLiteral], PropLiteral)$

Due to lack of space, we will not discuss the details and the implementation of the set of arguments; see [7] for details.

3.2 Carneades Argument Evaluation Structure

The main structure of the argumentation model is called a Carneades Argument Evaluation Structure (CAES):

▶ **Definition 1** (Carneades Argument Evaluation Structure (CAES))**.** A *Carneades Argument Evaluation Structure* (CAES) is a triple $\langle arguments, audience, standard\rangle$ where $arguments$ is an acyclic set of arguments, *audience* is an audience as defined below (Def. 2), and *standard* is a total function mapping each proposition to its specific proof standard.

The transliteration into Haskell is almost immediate

newtype $CAES = CAES\ (ArgSet, Audience, PropStandard)$

▶ **Definition 2** (Audience)**.** Let $\mathcal{L}$ be a propositional language. An *audience* is a tuple $\langle assumptions, weight\rangle$, where $assumptions \subset \mathcal{L}$ is a propositionally consistent set of literals (i.e., not containing both a literal and its negation) assumed to be acceptable by the audience and *weight* is a function mapping arguments to a real-valued weight in the range $[0, 1]$.

This definition is captured by the following Haskell definitions:

type $Audience = (Assumptions, ArgWeight)$
type $Assumptions = [PropLiteral]$
type $ArgWeight = Argument \to Weight$
type $Weight = Double$

Further, as each proposition is associated with a specific proof standard, we need a mapping from propositions to proof standards. A proof standard is a function that given a proposition p, aggregates arguments pro and con p and decides whether it is acceptable or not:

type $PropStandard = PropLiteral \to ProofStandard$
type $ProofStandard = PropLiteral \to CAES \to Bool$

The above definition of proof standard demonstrates that implementation in a typed language such as Haskell is a useful way of verifying definitions from argumentation theoretic models. Our implementation effort revealed that the original definition of proof standard as given in [12] could not be realised as stated, because proof standards in general not only depend on a set of arguments and the audience, but may need the whole CAES. However, due to space constraints we will omit the definition of specific proof standards.

3.3 Evaluation

Two concepts central to the evaluation (semantics) of a CAES are *applicability of arguments*, which arguments should be taken into account, and *acceptability of propositions*, which conclusions can be reached under the relevant proof standards, given the beliefs of a specific audience.

▶ **Definition 3** (Applicability of arguments)**.** Given a set of arguments and a set of assumptions (in an audience) in a CAES C, then an argument $a = \langle P, E, c\rangle$ is *applicable* iff

- $p \in P$ implies p is an assumption or $[\,\overline{p}$ is not an assumption and p is acceptable in $C\,]$ and
- $e \in E$ implies e is not an assumption and $[\,\overline{e}$ is an assumption or e is not acceptable in $C\,]$.

▶ **Definition 4** (Acceptability of propositions)**.** Given a CAES C, a proposition p is *acceptable* in C iff $(s\ p\ C)$ is $true$, where s is the proof standard for p.

The realisation of applicability and acceptability in Haskell is straightforward:

```
applicable :: Argument → CAES → Bool
applicable (Arg (prems, excns, _)) caes@(CAES (_, (assumptions, _), _))
  = and $ [(p ∈ assumptions) ∨ (p `acceptable` caes) | p ← prems]
          ++
          [(e ∈ assumptions) ↓ (e `acceptable` caes) | e ← excns ]
    where
      x ↓ y = ¬ (x ∨ y)
acceptable :: PropLiteral → CAES → Bool
acceptable c caes@(CAES (_, _, standard))
  = c `s` caes
  where s = standard c
```

4 Conclusions and future work

As we have seen, functional programming allows to realise structured argumentation models in such a way that the implementation is sufficiently close to the mathematical definitions to serve as specifications in their own right. Furthermore, by making the code publicly available and open source we improve the chances that implementation methods will progress more efficiently. For related work, researchers working in answer set programming have been working on extensions of ASP to make it more of a general purpose programming language. This has also allowed Charwat et al. to implement argument generation and visualisation for structured based approaches [5].

For future work, we are putting the formalisation methods discussed in Section 2.4 into practice, see also [8]. Thus, in addition to the discussed implementation of argumentation models using the described methods and tools, we want to verify the correctness of implementations and furthermore employ the same techniques for translations between argumentation models. Our ultimate goal is to have verified translations from structured argumentation models into efficiently implemented abstract argumentation models, resulting in a verified and efficient implementation method for structured argumentation models.

References

1 Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic framework for default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

2 Gerhard Brewka, Paul E. Dunne, and Stefan Woltran. Relating the semantics of abstract dialectical frameworks and standard AFs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 780–785, 2011.

3 Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 102–111. AAAI Press, 2010.

4 Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Implementing abstract argumentation - a survey. Technical Report DBAI-TR-2013-82, Vienna University of Technology, 2013.

5 Günther Charwat, Johannes Peter Wallner, and Stefan Woltran. Utilizing ASP for generating and visualizing argumentation frameworks. *CoRR*, abs/1301.1388, 2013.

6 Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

7 Bas van Gijzel and Henrik Nilsson. Haskell gets argumentative. In *Proceedings of the Symposium on Trends in Functional Programming (TFP 2012), LNCS 7829*, pages 215–230, St Andrews, UK, 2013. LNCS.

8 Bas van Gijzel and Henrik Nilsson. Towards a framework for the implementation and verification of translations between argumentation models. Draft Proceedings of The 25th symposium on Implementation and Application of Functional Languages, August 2013.

9 Bas van Gijzel and Henry Prakken. Relating Carneades with abstract argumentation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 1113–1119, 2011.

10 Bas van Gijzel and Henry Prakken. Relating Carneades with abstract argumentation via the ASPIC$^+$ framework for structured argumentation. *Argument & Computation*, 3(1):21–47, 2012.

11 Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-15):875–896, 2007.

12 Thomas F. Gordon and Douglas Walton. Proof burdens and standards. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 239–258. Springer US, 2009.

13 John Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, April 1989.

14 D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

15 Andres Löh. lhs2tex. `http://www.andres-loeh.de/lhs2tex/`. Accessed July 10, 2013.

16 Simon Marlow et al. Haskell 2010 language report. `http://www.haskell.org/onlinereport/haskell2010`, 2010.

17 Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1:93–124, 2010.

18 Guillermo R. Simari. A brief overview of research in argumentation systems. In *Proceedings of the 5th international conference on Scalable uncertainty management*, SUM'11, pages 81–95, Berlin, Heidelberg, 2011. Springer-Verlag.

Towards the Development of a Hybrid Parser for Natural Languages

Sardar F. Jaf[1] and Allan Ramsay[2]

1 **School of Computer Science, The University of Manchester**
2.46 Kilburn Building, Oxford Road, Manchester, M13 9PL, United Kingdom
sardar.jaf@manchester.ac.uk

2 **School of Computer Science, The University of Manchester**
2.21 Kilburn Building, Oxford Road, Manchester, M13 9PL, United Kingdom
allan.ramsay@manchester.ac.uk

Abstract

In order to understand natural languages, we have to be able to determine the relations between words, in other words we have to be able to 'parse' the input text. This is a difficult task, especially for Arabic, which has a number of properties that make it particularly difficult to handle.

There are two approaches to parsing natural languages: grammar-driven and data-driven. Each of these approaches poses its own set of problems, which we discuss in this paper. The goal of our work is to produce a hybrid parser, which retains the advantages of the data-driven approach but is guided by grammar rules in order to produce more accurate output. This work consists of two stages: the first stage is to develop a baseline data-driven parser, which is guided by a machine learning algorithm for establishing dependency relations between words. The second stage is to integrate grammar rules into the baseline parser. In this paper, we describe the first stage of our work, which is now implemented, and a number of experiments that have been conducted on this parser. We also discuss the result of these experiments and highlight the different factors that are affecting parsing speed and the correctness of the parser results.

1 Introduction

Processing human languages to determine the structural relations between words is called *parsing* in Computational Linguistics (CL) [1, p.63]. Parsing is one of the core components of many Natural Language Processing applications [2], such as: Machine Translation Systems, Tutoring and Speech Recognition Systems, Information Retrieval Systems, and Question and Answering Systems.

Producing a comprehensive parser is a challenging task due to language ambiguities [13], which is caused by such factors as multiple interpretations of words, flexibility of word order, and missing items. Hence, adequate parsing systems are often unavailable for natural languages, especially for languages with complex structures such as Arabic [18, p.82].

It is desirable that parsers have three main features - efficiency, robustness and accuracy. The efficiency of parsers is concerned with consuming as little time as possible, the robustness is for enhancing the system's ability to cope with agrammatical inputs, and the accuracy is required to ensure that the results produced are accurate. However, it is not possible to achieve all three features at once [7]. Some parsers have traded off accuracy for efficiency, while others sacrificed efficiency for robustness [18]. The goal of our work is to optimise speed and accuracy while maintaining a reasonable level of robustness. We aim to test our parser on Arabic because Arabic presents a number of challenges which make it hard to parse, and hence it will act as a rigorous test-bed for our approach.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 49–56

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Parsing natural language approaches

There are two different kinds of approaches to parsing natural languages: grammar-driven approaches, and data-driven approaches. In grammar-driven approaches, the parser depends on grammatical rules suitably specified in accordance with some linguistic theory. while in data-driven approaches, the parser mainly depends patterns extracted from preprocessed data, such as treebank data. In this section, we describe each of these approaches.

2.1 Grammar-driven approach

Grammar-driven parsing uses a formal grammar G, which defines a formal language $L(G)$ for an alphabet A. A grammar G is used as a system for generating strings over A; The language $L(G)$ that is defined by G is the set of all strings x that can be generated by G.

The assumption in grammar-driven parsing is that $L(G)$ is an approximation of the language L. However, the formal grammars that have been developed to date fail to meet this assumption [7]. Having said that, the principles behind grammar-driven approaches should not be neglected because the linguistic theories advancement may subsequently lead to better approximations, but, in the mean time it creates some practical problems for grammar-driven parsing.

Robustness in parsing natural languages is the capacity of parsers to analyse as many input strings as possible. A parser is considered robust if it can analyse a large proportion of the sentences of the language in question. One of the major problems associated with grammar-driven parsing is robustness. This problem occurs because some input strings in a given sentence may not exist in the formal language $L(G)$ that is defined by the grammar G. Generally, there are two kinds of robustness problem: (i) *coverage problem* and (ii) *robustness proper*.

The coverage problem normally occurs when an input string x is not part of the formal language $L(G)$ even though it is grammatically a legitimate sentence of L. Hence, at least in theory, it should exist in $L(G)$, because $L(G)$ is an approximation of L, but the sentence may not exist in $L(G)$ because it is not covered by G. On the other hand, robustness proper occurs when an input string x is understandable by the speaker of L but it is not part of the language L and so it should not exist in $L(G)$ either. Robustness proper actually occurs if a word is misspelled or if material is omitted or agreement constraints are violated.

The robustness problem can be solved by relaxing grammar constraints in parsers [4]. But, relaxing grammar constraints could result in many analyses becoming available for a given input text, hence it leads to the problem of *disambiguation*, which is a major problem for parsing natural language sentences, because applications that depend on the output of parsing systems typically require a small number of analyses (preferably just one analysis) for a given input text x. Having many analyses for a given x means that parsers will have to consume more time and resources exploring these analyses which create a problem with efficiency and also may result in selecting an incorrect analyses, which may affect accuracy.

The problem of robustness and disambiguating aggravates the problem of *accuracy*. Robustness attempts to produce analyses for input strings x that may not exist in $L(G)$ that is defined by G, while disambiguation insists on removing extra analyses that are assigned to x by G. These moves by any parsers may reduce the chance that an x that is part of a text is given the correct analysis by parsers. Hence, a joint optimisation between robustness, disambiguating and accuracy is necessary, or at least they are prioritised and the trade off between them is carefully chosen based on the nature of the parsing system.

Joint optimisation between robustness, disambiguating and accuracy triggers the problem

of *efficiency*. The problems with efficiency in grammar-driven approach depend mainly on the expressivity and the complexity of the formal language that is used for parser [7]. The most commonly used algorithms are at least N^3 in the length of the input text, which is daunting in situations where sentences may contain tens or even hundreds of words.

2.2 Data-driven parsing

In data-driven approaches, an inductive mechanism is used for mapping from input strings to output analyses. This mechanism, that is applied to a text sample $T_t = (x_1,...,x_n)$ from the language L to be analysed, makes the abstract problem of data-driven approach that is used to approximate text parsing a problem of inductive inference.

According to [7], data-driven parsers consist of three main components: (i) permissible analyses for sentences in the language L as defined by a formal model M. (ii) a sample of text $T_t = (x_1,...,x_n)$ from L with or without the correct analyses $A_t = (y_1,...,y_n)$, and (iii) actual analyses for the sentences $T = (x_1,...,x_n)$ in L is defined by an inductive inference scheme I, which is relative to model M and T_t and possibly A_t. Based on these components, a model M could represent a formal grammar G for restricting string representations to strings of the language L.

Training data, that is used in this approach, is a sample of text T_t. This could be raw data or an annotated treebank of the language L, where treebanks may or may not be annotated with representations satisfying the constraints of M. If the sample data is a treebank then a form of *supervised machine learning* is used for inductive learning because, according to the treebank annotation, the correct analyses of an input string x_i is in the sequence of analyses $A_t = (y_1,...,y_n)$. While *unsupervised machine learning* is used if the sample data is raw text because no sequence of analyses will exist in T.

Similarly to grammar-driven approaches, data-driven approaches are also based on the approximation that the formal language L is an approximation of the language L, but, this approximation is different in data-driven parsing because it is based on inductive inference from a finite sample $T_t = (x_1,...,x_n)$ to the infinite language L

The problem of robustness also exists in data-driven approach, robustness here depends on the formal model M properties as well as the inference scheme I which are used for processing new sentences. According to [7], in most existing data-driven parsers any input strings x are assigned at least one analysis, which means that data-driven parsers are highly robust. However, the extreme robustness of data-driven parsers means that they will assign analyses that are probably not in the language L.

Furthermore, the problem of disambiguation can be even more severe in data-driven parsers because the improved robustness is the result of extreme constraints relaxation. but, this is compensated by the fact that the inductive inference scheme I provides a mechanism for disambiguation, by associating a score with each analysis intended to reflect some optimality criterion, or, by implicitly maximising this criterion in a deterministic selection.

Regarding the problem of *efficiency*, it is argued that data-driven approaches is superior to grammar-driven approaches [7], but it is often at the expense of less accurate output [8]

3 Arabic

Ambiguity is a central problem in natural language parsing [11, 13]. Arabic contains many complexities and subtleties [10], which lead to even greater potential for ambiguities than is present with other languages. In the following sections we briefly highlight some of the main sources of ambiguities in Arabic.

3.1 Missing diacritics

Arabic diacritics are short strokes placed above or below consonants. There are three sets of diacritics: (i) Short vowels are symbols placed either above or below letters, such as /a/ sound as اَ, /u/ sound as اُ, or /i/ sound as اِ. (ii) Double case endings are also vowels and they suggest indefiniteness and are manifested in the form of case marking or in conjunction with case marking. these are placed on the final letter of a word, such as a /aN/ sound as in اً, /uN/ sound as in اٌ or /iN/ sound as in اٍ. (iii) Syllabification marks are placed above Arabic letters denoting the doubleing of the consonant, they are usually combined with short vowels. There are two types of syllabifications: (i) is called shadda written as a gemination marks as رَّ and (ii) is called sukun, which is a small circle as اْ, and it marks the boundaries between syllables, end of verbs, or it indicates that the word does not contain vowels.

Arabic texts without diacritics are ambiguous. Many words with different diacritic patterns appear identical in a diacriticless settings but they may have different syntactic roles [6]. e.g. the word علم *alam* can have many roles when diacritised, such as: noun as in عِلْمٌ *'ilmuN* "knowledge", transitive verb as in عُلِّمَ *'u-llima* "is taught" or intransitive verb as in عُلِمَ *'ulima* "is known". Written Modern Standard Arabic (MSA) generally omits diacritics, which leads to widespread lexical ambiguity of the kind shown [17].

3.2 Free word order

Arabic has a high degree of syntactic flexibility [10]. The canonical order of an Arabic sentence is VSO. But, a range of other word orders such as VOS, SVO and OVS are also possible [3], which is a source of ambiguities in Arabic [14, p.179]. It is not easy to distinguish between the nominative and accusative cases when word orders are changed, i.e. it is hard to identify the subject and object of a sentence, for example, in the sentence اَحمَد يَحتَرم عَلي *a.hmad ya.hatarm 'aly* "Ahmed respects Aly" it is clear that Ahmed is the subject in the sentence and Aly is the object. However, reordering the words in the same sentence as يَحتَرم أَحمَد عَلي *ya.htarm 'a.hmad 'aly* "respects Ahmed Aly" means that the subject could be either Ahmed or Aly. This results in structural ambiguity.

3.3 Arabic clitics

Clitics are morphemes [1] that possess the syntactic characteristics of a word, but, they are morphologically bound to other words [5]. Arabic clitics could be attached to the start or end of words, this often alters their formation, for example they could alter word types from noun to verbs, or even changes the verb type from transitive to intransitive [17]. For example, conjunctions in Arabic can often appear as clitics and modify Arabic verbs. For instance, the sentence وليَّهُم عَليٌ في المسآلة *wali-yyahum'alyuN fy Al mas'Ala* "Ali is the leader in their situation" where وَلِيَّهُمْ *walyahum* "their leader", which is a noun, is ambiguous because the letters و /w/ and ل /l/ could be clitics attached to the word لِيَّهُم *li-yyahum* "take charge" and can modify these words into verbs, as in the sentence وليَّهُم عَليٌ في المسآلة *wa li -yyahum 'alyuN fy Al mas'AlT* "and Ali to take charge of the situation", where the word is a verb.

[1] A morpheme is a small grammatical unit of a language.

3.4 Noun multi-functionality

It is difficult to define Arabic nouns in comparison to its verbs because they encompass a wide range of categories. One of the reasons that Arabic nouns create ambiguities is that some nouns are derived from verbs, and they can function as verbs sometimes [14]. e.g., البَحث *Alba.h_t* "search" can function as a noun as in اِستَمَرَّ التِلميذُ في البَحثِ لِلجَامِعَة *istama-rra Altilmy_du fy Alba.h_ti liljAmi'aT* "the student continued in his research for the university", and as a verb as in اِستَمَرَّ التِلميذُ في البَحثِ عَن الجَامِعَة *istama-rra Altilmy_du fy Alba.h_ti 'an AljAmi'aT* "the student continued searching for the university"

3.5 Arabic pro-drop

In pro-drop, the subject of a sentence could be omitted if the verb's agreement features are rich enough to recover its content [15]. Arabic verbs recover missing subjects by conjugating themselves to indicate the gender, number and person of the omitted pronoun subject [14]. Arabic pronouns may be omitted if the verb can recover them, as in, اكلت الدجاجة *Akalat Al dajAjT* "ate the chicken". The verb اكلت *Akalat* "ate" indicates that the missing subject is a singular, feminine and third person pronoun. In Arabic, verbs can be transitive and intransitive when a pronoun is dropped. It is not clear from the above sentence that the *NP* الدجاجة *Al dajAjT* "the chicken" following the verb اكلت *Akalat* "ate" is the subject. The sentence would mean *the chicken was eaten* and the verb اكلت *Akalat* "ate" is intransitive if the *NP* is the subject. But, the sentence would mean *she ate the chicken* and the verb اكلت *Akalat* "ate" is transitive if the *NP* is the object of the verb and the subject is an omitted pronoun (as "she"). Hence, due to pro-drops, parsers generate different structural analysis.

4 Work to date

We have implemented a dynamic programming version of shift-reduce parsing algorithm [1, p.368]. The dynamic programming algorithm stores partial parse analyses that are generated by the shift-reduce parsing algorithm. The shift-reduce parser has two data structures, queue and stack which contain items with the following features: (i) Part of Speech (POS), (ii) word form, (iii) word span within the sentence and (iv) words actual position in the sentence. Three operations are performed on queues and stacks: (i) *shift*, (ii) *left-reduce*, and (iii) *right-reduce*, where each of these operations results in a new queue and a new stack which are stored in a database as partial parse analyses. Shift operation moves the first token from the queue to the top of the stack producing a new queue and a new stack. *left-reduce* and *right-reduce* operations perform two main operations: First, a parent-daughter (dependency) relation between the first item on the queue and one of the items on the stack is determined. Second, the daughter (dependent) from the dependency relation is removed and the parent's start and end position is modified to create a span covering the position of the daughter within the parent's start and end position. Left-reduce operation makes a token from the stack a dependency parent of the token at the beginning of the queue. Right-reduce operation makes the token at the beginning of the queue a dependency parent of a token on the stack. These three operations continue until the queue is empty and a stack with one token is found in which the token's start and end position covers the whole sentence length in question, in which case the parse is considered successful. Otherwise, is unsuccessful.

We integrated an *oracle* in the parser for determining parser's action. We ask the oracle whether we should do a shift, left-reduce or right-reduce operation. If it suggests more than

one operations then we add them onto an agenda. The parser will then work through the agenda to explore the suggested actions specified in the agenda. The oracle determines the parser type, the parser becomes a grammar-driven parser if the oracle is a formal grammar, and it becomes a data-driven parser if the oracle is a set of inferred rules, which are extracted from a treebank. At this stage, the oracle is a set of inferred rules, which are extracted from a dependency treebank using a decision tree algorithm. Data-points for the decision tree algorithm are state:action pairs.

5 Preliminary results

We have conducted some preliminary tests on the the data-driven parser using a combination of techniques. The parser, by construction, is efficient and robust, because we provide it with shift, left-reduce or right-reduce action at each parse step, which deterministically lead to some analysis. However, the efficiency and robustness of the parser comes at the expense of its accuracy. We identified a number of factors that may potentially affect the accuracy, the main factors are: (i) the size of the queues and stacks used for parser training (where the Penn Arabic Treebank is used for training and testing the parser), (ii) the length of sentences, and (iii) the size of training data. As shown in Table 1, having one item on the queue affected the accuracy greatly, while variations in the stack size have less effects on the accuracy. The parser accuracy improved significantly when we trained it with queues containing more than one item. We also tested the parser on various sentence lengths. We divided the sentences in the test data into different sets of data size ranging from eight words to one hundred or more words. The highest accuracy is obtained with sentences that have less than twenty five words, as shown in Table 2. Using smaller training size than testing size also affects the accuracy. Larger training data produce better accuracy as can be seen from Table 3. The current state-of-the-art data-driven parser achieve %85 accuracy by combining a weighted combination of two state-of-the-art parsers (MaltParser and MSTParser) [12]. Our accuracy is below state-of-the-art because, at this preliminary stage, we included a limited number of features on that queue and stack which are used for parser training.

Table 1 Parser training with various number of items on queue and stack.

Queue items	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
Stack items	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Results (%)	25	25	24	10	57	57	57	57	60	60	60	60	60	61	61	61

Table 2 Parsing with various sentence length.

No. words in Sentences	8 to 25	26 to 50	51 to 75	76 to 100	over 100
Results (%)	66	59	59	58	57

Table 3 Parsing 500 sentences with training data ranging from 125 to 5000 sentences.

Training data	125	250	500	1000	2000	3000	4000	5000
Results (%)	55	57	58	59	61	60	60	60

5.1 Related work

There is an increasing interest for combining various parsing algorithms. Some work involved combining state-of-art dependency data-driven parsers, such as MaltParser [7] and MSTParser [18], while some other works focused on combing data-driven and grammar-driven parsers. The latter is the type of work that is more relevant to the work we present in this paper.

[16] combined a grammar-driven parser (XLE system), which is based on Lexical Functional Grammar (LFG), with a data-driven parser (MaltParser). In their approach, they supply a data-driven parser with outputs from a grammar-driven parser. The grammar-driven parser outputs phrase structured trees containing grammatical features. They convert the output of their XLE platform to dependency trees in order to have two parallel versions of the treebank: (i) a gold standard treebank, (ii) and a dependency treebank by converting the XLE system output which contains additional grammatical features. They extend the gold standard treebank with additional information from the corresponding LFG analyses. MaltPaser is then trained on the enhanced gold standard treebank. Their results showed a small improvement in accuracy when applied to English and German.

A similar work in this area is conducted by [9]. They constrain a Head-driven Phrase Structure Grammar (HPSG) parser with outputs from a data-driven parser. HPSG parsers use a small number of schemas for explaining general construction rules, and a large number of lexical entries for expressing word-specific syntactic and semantic constraints. HPSG parse trees are converted to Context Free Grammar style (CFG-style) trees and a dependency treebank is then extracted from the CFG-style trees. the dependency treebank is used for training a dependency data-driven parser, such as MaltPaser and MSTParser. Outputs from data-driven parsers are used to constrain the HPSG parser. During HPSG parsing process, the lexical head of each partial parse tree is stored and in each schema application the head child is determined. Having such information about the head child and the lexical head, the dependency produced by the schema application is identified and whether the schema application violates the dependencies in the dependency treebank is checked. The HPSG parser is forced to produce parse trees that are consistent with the dependency trees. This approach is tested on English and some improvements in accuracy was achieved.

5.2 Next stage

The next stage is to implement a hybrid parser by integrating grammatical rules into this parser. The aim is to constrain our data-drive parser with features from grammar-driven approaches. In order to produce a hybrid parser, we make the oracle a weighted combination of grammar W and decision tree D. We fix D to be 1 (or 1 times whatever probability we can extract from it). If $W = 0$ then the parser is a data-driven parser. If $W = N{+}1$, where N is the length of sentence, then the parser becomes a grammar-driven parser. Intermediate values produce combinations. Low value for W will be fairly fast but prone to producing non-conforming trees, high value for W will be slow but trees will tend to be legal.

6 Conclusion

Problems associated with using grammar-driven approaches and data-driven approaches are discussed in this paper. The main structural complexities of Arabic are identified and described in section 3. The first stage of our approach to hybrid parsing is explained and the next stage for full hybrid parsing implementation is established. Preliminary results for the parser is included in section 5. The parser is tested on Arabic because it is a complex

language, compare to some other languages, hence it provides a rigorous test-bed. Finally, two various related works in hybrid parsing approaches for natural language processing is identified and briefly described.

References

1 Alfred, V., Aho and Jeffery, D., Ullman. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentic-Hall, 1972.

2 Ali, Farghaly and Khaled, Shaalan. Arabic Natural Language Processing: Challenges and Solutions. *ACM Computing Surveys*, 8(4):1–22, 2009.

3 Allan, Ramsay and Hanady, Mansour. Local Constraints on Arabic Word Order. In *Proceedings of the 5th international conference on Advances in Natural Language Processing*, FinTAL'06, pages 447–457, Berlin, Heidelberg, 2006. Springer-Verlag.

4 Christa, Samuelsson and Mats, Wirèn. *Parsing techniques.* Marcel Dekker, 2000.

5 David, Crystal. *A First Dictionary of Linguistics and Phonetics.* Deutsch, London, 1980.

6 Imed, Zitouni and Jaffery, S., Sorensen and Ruhi, Sarikaya. Maximum Entropy Based Restoration of Arabic Diacritics. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 577–584, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

7 Joakim, Nivre. Inductive Dependency Parsing. Springer, 2006.

8 Ronald M. Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell III, Alexander Vasserman, and Crouch Richard. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of Human Langauge Technology and the Conference of the North American Chapter of teh Association for Computational Linguistics (HLT-NAACL*, pages 97–104, 2004.

9 Kenji, Sagae and Yusuke, Miyao. Hpsg parsing with shallow dependency constraints. In *In Proc. ACL 2007*, 2007.

10 Kevin, Daimi. Identifying Syntactic Ambiguities in Single-parse Arabic Sentence. 35(3):333–349, 2001.

11 Marlyse, Baptista. On the Nature of Pro-drop in Capeverdean Creole. 5:3–17, 1995.

12 Maytham, Alabbas and Allan, Ramsay. Evaluation of combining data-driven dependency parsers for arabic. In *Proceedings of the 5th Language & Technology Conference Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 546–550, 2011.

13 Michael, Collins. Head-Driven Statistical Models for Natural Language Parsing. *Comput. Linguist.*, 29(4):589–637, 2003.

14 Mohammed, A., Attia. *Handling Arabic Morphological and Syntactic Ambiguities within the LFG Framework with a View to Machine Translation.* PhD Thesis, School of Languages, Linguistics and Cultures, Manchester University, 2008.

15 Noam, Chomsky. *Lectures on Goverment and Binding.* Dordrecht: Foris, 1981.

16 Øvrelid, Lilja and Kuhn, Jonas and Spreyer, Kathrin. Improving data-driven dependency parsing using large-scale lfg grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 37–40, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

17 Rani, Nelken and Stuart, M., Shieber. Arabic Diacritization Using Weighted Finite-State Transducers. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, Semitic '05, pages 79–86, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

18 Ryan, MacDonald. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing.* PhD Thesis, Computer and Information Science, the University of Pennsylvania, 2006.

Improving the quality of APIs through the analysis of software crash reports

Maria Kechagia, Dimitris Mitropoulos, and Diomidis Spinellis

Athens University of Economics and Business
Department of Management Science and Technology
{mkechagia, dimitro, dds}@aueb.gr

Abstract

Modern programs depend on APIs to implement a significant part of their functionality. Apart from the way developers use APIs to build their software, the stability of these programs relies on the APIs design and implementation. In this work, we evaluate the reliability of APIs, by examining software telemetry data, in the form of stack traces, coming from Android application crashes. We got 4.9 GB worth of crash data that thousands of applications send to a centralized crash report management service. We processed that data to extract approximately a million stack traces, stitching together parts of chained exceptions, and established heuristic rules to draw the border between applications and API calls. We examined 80% of the stack traces to map the space of the most common application failure reasons. Our findings show that the top ones can be attributed to memory exhaustion, race conditions or deadlocks, and missing or corrupt resources. At the same time, a significant number of our stack traces (over 10%) remains unclassified due to generic unchecked exceptions, which do not highlight the problems that lead to crashes. Finally, given the classes of crash causes we found, we argue that API design and implementation improvements, such as specific exceptions, non-blocking algorithms, and default resources, can eliminate common failures.

1998 ACM Subject Classification D.2.2 Design Tools and Techniques, D.2.5 Testing and Debugging, D.2.7 Distribution, Maintenance, and Enhancement

Keywords and phrases application programming interfaces, mobile applications, crash reports, stack traces

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.57

1 Introduction

Many modern applications use Application Programming Interfaces (APIs) to build their basic functionalities. The stability of these applications depends not only on the use of the APIs by developers, but, also, on the API design and implementation itself.

Even though the software engineering literature encounters works related to software development practices [7], metrics [6], and bug report analysis [10], there are limited studies regarding APIs. We have mainly found sources regarding the usability of APIs [8], [9] and general design practices [4], [2]. Therefore, there is a demand for studies on the assessment of APIs.

In this work, we report how we used software telemetry data, in the form of stack traces, coming from Android application crashes, to analyze their causes and evaluate the reliability of the used APIs. First, we got a 4.9 GB data dump of crash reports from several mobile applications. Then, we processed that data to get an amount of a million Java stack traces in an appropriate form for our analysis. Finally, we applied heuristic rules to draw the border between applications and API calls. This helped us to locate problematic calls to API

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 57–64

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

methods and investigate the reasons that these API deficiencies lead to applications crashes. In addition, knowing the crash causes map of our sample's applications, we were able to argue about related API design and implementation recommendations.

We chose to focus on the study of API crashes for a number of reasons. First, crashes that could have been avoided through a better API land on the hands of application builders. These builders can fix their applications on a case-by-case basis. Thus, locating weaknesses in the APIs and improving their design or implementation can ensure the stability of the thousands of applications that use them. Finally, the fact that most APIs are available as open source software makes them a valuable ground for research.

In addition, we chose the Android platform as the subject of our study because of its popularity, diversity, and availability. Specifically, more than 800 million devices use the Android platform and 700,000 applications are written for Android. In addition, the Android API is quite large (3,000 classes and 300 packages) for examination and its interfaces are open source.

In the rest of this work, we first outline the methods we used (Section 2). In Section 3, we discuss the crash categories we found, and make API recommendations. In Section 4, we present the threats to validity of our study, and we end up with our conclusions and future work in Section 5.

2 Methodology

Our methodology involves data collection, cleaning, processing, and analysis. First, we got our data set and we conformed the stack traces to a certain format for analysis. Then, we applied heuristic rules to the stack traces to extract from each a representative triplet—**signature** hereafter, related to the crash cause. We sorted the signatures based on the times they appear in the stack traces, and we examined the top 600 ones (80% of the total population) to investigate the reasons behind the application failures. Finally, we categorized the crash causes we found into main classes and we made related API recommendations.

2.1 Data Origin

The subject of our empirical study consists of Java stack traces, coming from Android application crashes, collected through a centralized crash report management service.

Android mobile phones are embedded devices that use the Linux operating system and host applications. Here, we briefly discuss an overview of the Android framework. In the bottom layer, there is the Linux kernel, which is the border between the device and the software. It provides services such as memory management, networking, and power management. In the middle layer, there is the Dalvik process virtual machine (VM) for the running of several applications on the system and the Java Native Interface (JNI) that is used to perform calls from Java code into native code. Finally, on the top layer, there are several Java classes coming from: 1) basic applications (contacts, browser, phone), 2) third-party applications, and 3) the Java Platform (J2SE). The methods of these classes are used for the development of Android applications and consist subject of our study.

The provider of our data set is the BugSense Inc., a privately held company, founded in 2011, and based in San Francisco. The aim of BugSense is to provide error reports and analytics regarding the performance and quality of mobile applications. Our sample comes to 2,042,700 crash reports, collected in real time from the 13th of January of 2012 to the 11th of April of 2012, from 4,618 distinct applications. The examined Android API refers to versions from 1.0.0 to 4.1.1.

2.2 Data Cleaning

In order to conduct our analysis, we first needed to clean our data. From our initial sample, we only kept Java stack traces from Android applications. For this, we wrote a program in Python and parsed our data set. Specifically, by using regular expressions, we checked the format validity of the stack traces, based on the `printStackTrace()` method, from the `Throwable`[1] Java class. Thus, from our initial sample, we concluded on 901,274 well-formed Java stack traces. Listing 1 shows a representative example from the `Throwable` documentation. In addition, we transformed each stack trace for further analysis. We reversed each exception level sequence of call methods and joined the levels at the common methods. Then, the final chain of Listing 1 would be `.main.a.b.c`.

Listing 1 Throwable stack trace.

```
HighLevelException: MidLevelException:
        at Junk.a(Junk.java:13)
        at Junk.main(Junk.java:4)
Caused by: MidLevelException:
        at Junk.c(Junk.java:23)
        at Junk.b(Junk.java:17)
        at Junk.a(Junk.java:11)
        ... 1 more
```

2.3 Identification of Risky API calls

Isolating calls to arbitrary APIs within stack traces of unknown application code called in diverse ways from a larger framework is not trivial. In general, a stack trace of method calls from the Android framework F leading to an exception E, possibly through an application A and an API I. This can be expressed by the following regular expression.

$$((F + (A + I*)*)|(F * (A + I*)+))E$$

This expression reflects several cases in which an exception can occur. For instance consider:

- Within the Android framework: $F + E$
- Within the application: $F * A + E$
- When the application calls an API: $F * A + I + E$
- Within an API-registered application callback: $F * (A + I + A+) + E$
- When an API-registered application callback calls an API: $F * (A + I+) + E$

To locate the API calls that lead to application crashes we had to locate the last instance of an AI pair. We had however no *a priori* knowledge of the methods that belong to the sets F, A, and I. Thus, we used heuristics to determine them. In particular, we constructed from the stack traces n-tuples of length 1–15 anchored to the left hand side of the stack trace and determined their times of occurrence. Looking at the most common ones, we manually established the name space of the Android framework's methods. The sixth most common n-tuple we found was the following:

[1] `http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html`

```
dalvik.system.NativeStart.main
com.android.internal.os.ZygoteInit.main
com.android.internal.os.ZygoteInit$\
  MethodAndArgsCaller.run
java.lang.reflect.Method.invoke
java.lang.reflect.Method.invokeNative
android.app.ActivityThread.main
```

From this 6-tuple we deduced that the framework calls applications through methods that belong to the packages `dalvik.*`, `com.android.*`, `java.*`, and `android.*`. In addition, we searched for other common n-tuples from third parties to fill the framework's name space. For instance, these are the top ones: `com.badlogic.gdx.backends.android.*` and `org.cocos2d.*`.

Knowing the application's name space, we searched the stack traces *backwards* (from the RHS to the LHS) to locate the first place where an application's method called a method that did not belong to its name space (an *AI* sequence). This was, by definition, a call to an API method. In Listing 2, the interesting API call is that to the `setContentView` method.

Listing 2 Exceptional sequence.

```
com.example.Serialize$Looper.run
android.os.Looper.loop
android.os.Handler.dispatchMessage
com.example.SerializeHandler.onMessage
com.example.app.Activity$1.work
android.app.Activity.setContentView
```

Finally, from the stack traces we extracted for further analysis signatures representing the API method (e.g. `android.app.Activity.setContentView`), the exception reported by the API method (e.g. `android.view.inflateException`), and the root exception that triggered the application crash—the exception at the bottom of the stack (e.g. `java.lang.NullPointerException`). Each signature represents a way in which an API call can fail. Thus, one signature can be associated with many different stack traces and reflects the main cause of a crash. We used the signatures in order to group our data and as a guide for studying the reason of an application failure behind the thrown exceptions.

3 Crash Categories and API Recommendations

Further analyzing our data set, we wanted to see *why* application crashes occur and *what* API deficiencies are responsible for them. To achieve this, we examined the signatures we extracted from the stack traces and we identified major classes of crash causes. In particular, we sorted the signatures according to their number of occurrence and we got the top 600 ones (80% of the stack traces) for analysis. For each signature, we identified the reason of its application failure, and allocated the signature to a broad crash cause category (Figure 1). For the signatures with specific exceptions (e.g. `OutOfMemoryError` and `OutOfBoundsException`) it was easy for us to understand the problems. However, for these with generic unchecked exceptions (e.g. `RuntimeException`, `NullPointerException` and `IllegalArgumentException`) the reason of the execution failure was not clear. Thus, we needed to search in the Android API reference and consulting sites (*stackoverflow)*[2] to

[2] `http://stackoverflow.com`. All sites were accessed on the 20th of July, 2013.

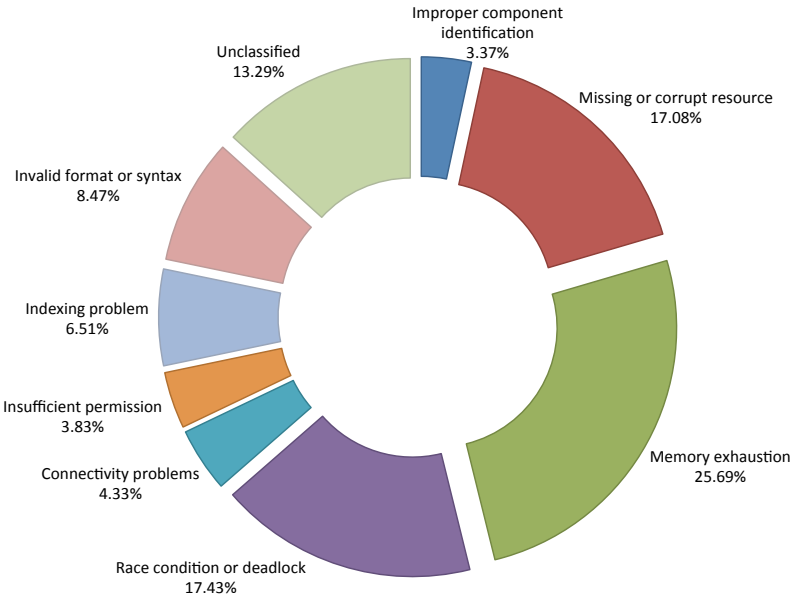


Figure 1 Causes of API-related crashes from top 600 stack traces (80% of total crashes).

reveal the real crash causes. Table 1 presents each crash cause category giving examples of signatures allocated to them and illustrates related API recommendations. Following we discuss each crash category and provide indicative API design and implementation choices.

Memory exhaustion is the most common application crash cause. This was a result we expected, as mobile devices can have constrained memory and developers are seldom aware of the amount of the available memory. Table 1 shows a characteristic example of this category related to a failed import operation for a bitmap. In order to decrease the number of this category's crashes, the API can include an interface for the adaptation of memory consuming resources, so that they can fit in the memory. For instance, if an image cannot be loaded, the developer could sample it first. Moreover, the API can restrict the use of cache structures that trigger memory leaks. Finally, the API can permit the use of file formats that can consume less memory (vector graphics).

Race condition or deadlock is another significant cause of application crashes. This category contains signatures related to: a. database deadlocks, b. race conditions in asynchronous tasks (see Table 1), c. abnormal execution of the lifecycle of an activity, and d. synchronization issues with iterators. To eliminate these crashes, the API should provide non-blocking primitives. Also, developers can catch these problems by using profiling (Traceview[3] and Jinsight[4]) and testing tools [1].

Missing or corrupt resource cause refers, also, to a great number of crashes. In this category, we have added signatures that imply the absence of a resource or inability of the system to decode a resource (see Table 1). We refer to external resources, such as an image or an audio file, and not application components (activities, services, broadcast receivers, and content providers). Crashes because of missing resources can be avoided if the API includes default resources (e.g. layouts). In addition, we found that some exceptions related to this category are unclear (`NullPointerException`). Thus, it is not easy for the developers

[3] `http://developer.android.com/tools/debugging/debugging-tracing.html`
[4] `http://www-03.ibm.com/systems/z/os/zos/features/unix/tools/jinsightlive.html`

Table 1 Categories and Recommendations.

Categories	Signatures	Recommendations

Memory Exhaustion	`android.app.Activity.setContentView`
	`android.view.InflateException`
	`java.lang.OutOfMemoryError`
		Resource auto-resize interface
		Restricted use of cache structures (e.g. LruCache)
		Use of cheap file formats (vector graphics)

Race Condition or Deadlock	`android.os.AsyncTask.execute`
	`java.util.concurrent.RejectedExecutionException`
	`java.util.concurrent.RejectedExecutionException`
		Non-blocking algorithms
		Specific exceptions

Missing or Corrupt Resource	`android.app.Activity.setContentView`
	`android.view.InflateException`
	`java.io.FileNotFoundException`
		Default resources
		Specific exceptions

Improper Component Identification	`android.app.Activity.startActivity`
	`android.content.ActivityNotFoundException`
	`android.content.ActivityNotFoundException`
		Useful IDs
		Type Checking

Insufficient Permission	`android.app.Activity.startActivity`
	`java.lang.SecurityException`
	`java.lang.SecurityException`
		Clear documentation
		Specific exceptions

Invalid Format or Syntax	`android.database.sqlite.SQLiteDatabase.execSQL`
	`android.database.sqlite.SQLiteException`
	`android.database.sqlite.SQLiteException`
		Interface for queries on collections (e.g. JQL)

Indexing Problem	`java.util.ArrayList.get`
	`java.lang.IndexOutOfBoundsException`
	`java.lang.IndexOutOfBoundsException`
		Error-free arguments (iterators)
		Error ignorance (in loop conditions)

Connectivity Problems	`org.apache.http.impl.client.AbstractHttpClient.execute`
	`org.apache.http.NoHttpResponseException`
	`org.apache.http.NoHttpResponseException`
		User menu (1. wait, 2. new provider, 3. pause, 4. terminate)

Unclassified	`android.hardware.Camera.open`
	`java.lang.RuntimeException`
	`java.lang.RuntimeException`
		Clear documentation
		Specific exceptions

to understand where a crash comes from. This means that the API should offer specific exceptions regarding problematic resources.

Improper component identification category includes signatures that indicate crashes due to either undeclared components or system's failure to locate a suitable component for a specific task. Crashes of this category can occur because of wrong declaration of the application components (activity, service, broadcast receiver, and content provider). To prevent such crashes, the API can use more meaningful component codes (easy remembered) and appropriate type checks.

Insufficient permission category covers signatures related to crashes because of missing or incorrect activity permissions. Table 1 shows a representative example. The activity cannot start, as the Intent object, which should be passed to the system, has not got the right permissions (for another device to be eligible to receive a message). Specific exceptions and clear documentation provided by the API can eliminate such problems. Static checking tools, also, can help in the early location of permission issues [3].

Invalid format or syntax category refers to crashes due to erroneous method inputs. Specifically, the signatures that belong here imply format problems and invalid syntax of SQL queries. For instance, the corresponding exception in Table 1 reflects that the signature is related to a wrong SQL query syntax. In order to avoid such problems, the API can include an interface for queries on collections (e.g. JQL)[5]. Static checking tools can, also, eliminate

[5] `http://homepages.ecs.vuw.ac.nz/~djp/jql/`

these crashes (consider the *lint*[6] tool).

An **Indexing problem** can be caused by invalid loop conditions and inappropriate structures (see Table 1). Crashes due to these problems can be avoided with the use of error-free arguments (e.g. implicit loops and integer indices), as well as error ignorance (in case a threshold is greater than the size of a list). Also, static checking can solve such issues (consider FindBugs [5]).

Connectivity problems cover signatures associated with networking exceptions (see Table 1). To prevent such cases, the system instead of throwing exceptions can provide the user with a user menu for next actions, such as: 1) wait, 2) choose a new network provider, 3) pause the application, 4) terminate the application. Then, the user has to choose one of these options, and the system can proceed accordingly. Other possible solutions include stress tests and notifications from the system (via monitoring of the network activity).

Unclassified signatures do not give clear information about the real causes of their crashes. For instance, consider a representative example in Table 1 that reflects a crash where the camera cannot be opened. This occurs either because another application is using the camera or because the application has not got the permission to use the camera. However, the `RuntimeException` is generic for one to understand whether the crash cause is a race condition or an insufficient permission. We argue that such exceptions consist an API design problem, and there is a demand for more specific exceptions.

4 Threats to validity

In this section, we discuss the limitations of our study. Internal validity refers to the implications of the method used for the analysis of the stack traces. While, external validity aims to ensure that the findings of our empirical study can be generalized for other samples, too.

4.1 Internal validity

As we had no *a priori* knowledge of the methods that belong to the Android framework, we used heuristics to determine them. We sorted the n-tuples, by their frequency, and looked at the six most common ones to manually establish the name space of the Android framework's methods. In addition, we manually examined other common n-tuples to find application-specific methods. Therefore, although we identified the Android framework's methods that are used to call applications, we may have missed the less common ones, especially from third-party applications.

4.2 External validity

We argue that our findings can be representative of a large population for a number of reasons. First, we believe that for another Android sample we will possibly get the same results. To support this, we examined 600 signatures (rest 20% of the stack traces) through a random sample from our data set, and we validated our crash cause categories against our original data set. Second, we argue that our categories could be the same for another platform because of: 1) the amount of the crashes and applications we examined, 2) the fact that Android runs on a diversity of devices, 3) the size of Android's API, and 4) the generic nature of our categories. We have, however, to validate our results by examining data from platforms, such as iOS and Windows mobile.

[6] `http://developer.android.com/tools/help/lint.html`

5 Conclusions and Future Work

In this work, we presented an analysis of Java stack traces from Android application crashes and an investigation of the causes behind execution failures. As future work, we aim to analyze crash reports from other operating systems such as the iOS and Windows for mobile devices. In addition, we aim to combine the crash reports with other metadata related to specific devices and demographic data. Finally, we work toward to the examination of more stack traces from our sample and the validation of our categories, as well as the automation of our classification method.

Acknowledgements We would like to thank the BugSense Inc., and more specifically its founders Panos Papadopoulos and John Vlachogiannis. Also, we want to thank Panos Louridas, Georgios Gousios, Marios Fragkoulis, and Vassilios Karakoidas for their internal reviews.

This research has been co-financed by the European Union (European Social Fund — ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) — Research Funding Program: Thalis — Athens University of Economics and Business — Software Engineering Research Platform.

References

1 Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 258–261, New York, NY, USA, 2012. ACM.

2 Joshua Bloch. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 506–507, New York, NY, USA, 2006. ACM.

3 Patrick P.F. Chan, Lucas C.K. Hui, and S. M. Yiu. DroidChecker: analyzing Android applications for capability leak. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 125–136, New York, NY, USA, 2012. ACM.

4 Michi Henning. API design matters. *Commun. ACM*, 52(5):46–56, May 2009.

5 David Hovemeyer and William Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, December 2004.

6 Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

7 Steve McConnell. *Code Complete, Second Edition*, pages 133–143. Microsoft Press, Redmond, WA, USA, 2004.

8 Martin P. Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

9 J. Stylos and Carnegie Mellon University. *Making APIs More Usable with Improved API Designs, Documentation and Tools*. Carnegie Mellon University, 2009.

10 Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. Bug characteristics in open source software. *Empirical Software Engineering*, pages 1–41, 2013.

Fast Implementation of the Scalable Video Coding Extension of the H.264/AVC Standard*

Xin Lu and Graham R. Martin

Department of Computer Science
University of Warwick, Coventry
CV4 7AL, United Kingdom
{xin, grm}@dcs.warwick.ac.uk

Abstract

In order to improve coding efficiency in the scalable extension of H.264/AVC, an inter-layer prediction mechanism is incorporated. This exploits as much lower layer information as possible to inform the process of coding the enhancement layer(s). However it also greatly increases the computational complexity. In this paper, a fast mode decision algorithm for efficient implementation of the SVC encoder is described. The proposed algorithm not only considers inter-layer correlation but also fully exploits both spatial and temporal correlation as well as an assessment of macroblock texture. All of these factors are organised within a hierarchical structure in the mode decision process. At each level of the structure, different strategies are implemented to eliminate inappropriate candidate modes. Simulation results show that the proposed algorithm reduces encoding time by up to 85% compared with the JSVM 9.18 implementation. This is achieved without any noticeable degradation in rate distortion.

1998 ACM Subject Classification I.4.2 Compression (Coding), E.4 Coding and Information Theory

Keywords and phrases Fast mode selection, Inter-layer prediction, Scalable Video Coding (SVC), SVC extension of H.264/AVC.

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.65

1 Introduction

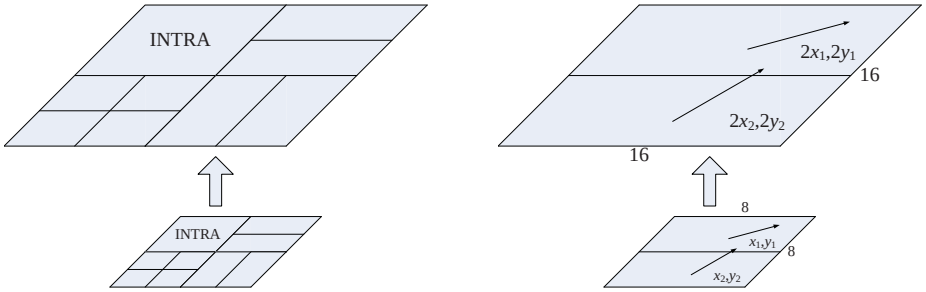
The Scalable Video Coding (SVC) extension of H.264, also known as MPEG-4 part 10 Advanced Video Coding Amendment 3 [2], is the result of joint standardisation work by ISO/IEC JTC1 SC29/WG11 (MPEG) and ITU-T. SVC enables a video sequence to be decoded fully or partially with variable quality, resolution and frame rate depending on the available network bandwidth or application requirements [10].

The coding mode decision process in the enhancement layer(s) requires an extremely large amount of computation. It is observed that this process dominates the encoding time in H.264/SVC. This is due to the application of many time consuming encoding tools. For instance, rate distortion optimisation and inter-layer prediction are involved in the mode decision process. Evaluation results show that the mode decision process in the enhancement layers accounts for 90% of the total computational requirement [3], and the encoding time of the enhancement layer is 10 times more than that of the base layer. In the rate distortion optimised mode decision process, more candidate partition modes are involved in SVC than any previous video coding standard. In SVC, all the encoding tools of H.264/AVC have been

* A longer version of this paper appeared in IEEE T-CSVT [8].

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 65–72

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Partition prediction. **(b)** Motion prediction.

Figure 1 Inter-layer prediction.

inherited [9], and these are supplemented by additional tools to support scalability. SVC uses a layer-based scheme to provide spatial and quality scalability [10], and in order to improve coding efficiency, it employs a mechanism to reuse the coded lower layer data for encoding the corresponding enhancement layer. This is referred to as inter-layer prediction [11]. In this, a new block mode is introduced, a base layer skip (BLSKIP), which applies three types of coding tools including inter-layer texture prediction, inter-layer motion prediction and inter-layer residual prediction, as shown in Figure 1. There are eight available macroblock modes for inter prediction in H.264/AVC, namely MODE_SKIP, MODE_16×16, MODE_16×8, MODE_8×16, MODE_8×8, MODE_8×4, MODE_4×8, MODE_4×4 (Figure 2), and two modes for intra prediction, INTRA_4×4 and INTRA_16×16 [13]. For an enhancement layer in SVC, all the modes concerned with inter prediction, intra prediction and inter layer prediction are evaluated, and the mode with the minimum rate distortion (RD) cost is selected as the best mode for the current macroblock. The rate distortion function for a block ω_k is given by

$$J(\omega_k, \hat{\omega}_k, \text{MODE}|\text{Qp}) = D(\omega_k, \hat{\omega}_k, \text{MODE}|\text{Qp}) + \lambda(\text{Qp}) \times R(\omega_k, \hat{\omega}_k, \text{MODE}|\text{Qp}) \quad (1)$$

where ω_k is an original macroblock at time k, and $\hat{\omega}_k$ is the corresponding constructed block. Qp is the quantisation parameter [12]. R represents the number of bits, and D denotes

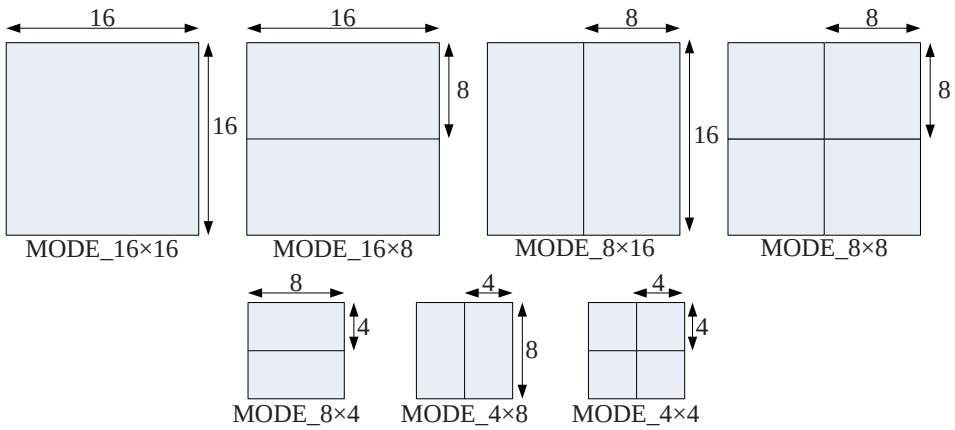


Figure 2 Block modes for inter-frame prediction in H.264/AVC.

a distortion measure. When exhaustive mode selection is used, the rate distortion cost must be evaluated for all modes in order to decide which mode should be employed. A set of time consuming encoding tools are incorporated in SVC as well as in H.264/AVC, for instance, bi-directional motion prediction, quarter-pixel precision motion estimation, and motion compensation using multiple reference blocks. In addition, the inter-layer prediction tools of SVC also incur excessive computational cost. In particular, inter-layer residual prediction doubles the computational complexity of the mode decision process [3]. Inter-layer motion prediction also results in a significant increase in computational complexity.

Several algorithms have been suggested to reduce the computational complexity of SVC. In [5], Kim *et al.* used the RD cost of the base layer skip mode and the information in the base layer to categorise macroblocks into different groups. The algorithm then reduces the number of candidate modes for the enhancement layer. The algorithm's performance is highly dependent on a constant K which determines the time saving and reconstructed picture quality. In [7], depending on the mode distribution relationship between the base layer and enhancement layers, Li *et al.* reduced the number of candidate modes in the enhancement layer according to the best mode in the corresponding position in the base layer. However, this method provides poor results when the correlation between base layer and enhancement layer is weak. Goh *et al.* [4] proposed an algorithm that makes use of the relationship between current macroblocks and their neighbours to decrease the encoding time. Nevertheless, for fast changing video sequences, it results in expected bit-rate degradation. In [16], Zhao *et al.* utilised the encoding mode of the base layer to initialise the candidate mode list of the enhancement layer, thus saving the overall encoding time. However the mode relationship between the macroblock and its neighbours is not investigated, and the time saving can be further improved.

In this paper we suggest a fast mode decision approach to alleviate the computational complexity of the SVC encoder without any significant loss of compression performance and coding efficiency. The remainder of this paper is organised as follows. Section 2 discusses the formulation of the proposed algorithm, and the overall structure is described in Section 3. Extensive experimental results are presented in Section 4 and conclusions are given in Section 5.

2 Observations and Analysis

The proposed algorithm is formulated as a hierarchical application of knowledge gained from previously processed information and the inherent content of the video being encoded.

2.1 Mode Correlation between Base Layer and Enhancement Layer

In order to support spatial and quality scalability, SVC adopts a multi-layer coding approach. To improve coding efficiency, a new prediction mechanism referred to as inter-layer prediction was incorporated into the standard. The purpose of using inter-layer prediction tools is to exploit as much lower layer information as possible for improving the coding efficiency of the enhancement layer.

In general, the base layer is first encoded independently, followed by the enhancement layer. As the input video of the different layers is generated from the same original video source but at different spatial resolution, the picture content is highly correlated [15]. Specifically, the prediction mode and motion vectors of the enhancement layer are strongly correlated with those of the base layer. Consequently, by exploiting the mode information of the

Table 1 % Mode Correlation between Base Layer and Corresponding Enhancement Layer.

Sequence	Qp				
	24	28	32	36	40
Bus	40.09	47.30	53.16	58.61	65.71
Foreman	42.90	53.97	61.04	69.14	79.94
Mobile	49.92	57.70	63.57	65.44	70.35
Mother-Daughter	78.81	85.40	90.37	95.03	97.64

corresponding macroblock in the base layer, the computational complexity of the mode decision process in the enhancement layer can be reduced significantly.

To illustrate the mode relationship between the base layer and the enhancement layers and to justify our proposed algorithm, we analysed the probability of macroblocks in the enhancement layers being encoded as MODE_SKIP when the mode of the co-located macroblock in the base layer is also MODE_SKIP. Table 1 shows the mode correlation between the base layer and the enhancement layer as defined in equation (2). We examined four video sequences with different degrees of activity and detail. The statistics were collected from the first 90 frames of each video sequence.

$$MC_{IL} = \frac{MB_{B\&E\_SKIP}}{MB_{E\_SKIP}} \times 100\% \tag{2}$$

where $MB_{B\&E\_SKIP}$ is the number of macroblocks predicted as MODE_SKIP in the base layer when the corresponding macroblock in the enhancement layer is MODE_SKIP too. MB_{E_SKIP} is the number of MODE_SKIP macroblocks in the enhancement layer.

From Table 1, we deduce that if the best mode for the macroblock in the base layer is MODE_SKIP, the corresponding macroblock mode in the enhancement layer is very likely to be MODE_SKIP as well. This is true regardless of video sequence.

2.2 Mode Correlation between Macroblock and its Neighbours

Besides the correlation between layers, there is also a significant dependency of neighbouring macroblocks in the enhancement layer. For a majority of video sequences, MODE_SKIP macroblocks tend to occur in clusters, such as a patch of static background. Consequently there is a high possibility that the best mode for the current macroblock is similar to that of its neighbours, that is, coded macroblocks which are located to the above and to the left of the current macroblock.

In order to reveal the mode dependency between macroblocks, we measured the probability that the current macroblock is coded as MODE_SKIP, if one or both of the neighbouring macroblocks are also coded MODE_SKIP.

Equation (3) denotes the mode correlation between the current macroblock and its neighbours.

$$MC_{SN} = \frac{MB_{C\&N\_SKIP}}{MB_{C\_SKIP}} \times 100\% \tag{3}$$

where MC_{SN} is the mode correlation between a macroblock and its spatial neighbours. $MB_{C\&N\_SKIP}$ is the number of the macroblocks which are predicted MODE_SKIP when the macroblock's neighbours in the enhancement layer are MODE_SKIP as well. MB_{C_SKIP} is the number of MODE_SKIP macroblocks in the enhancement layer. Table 2 shows the mode correlation between a macroblock and its spatial neighbours as defined in equation (3).

From the observations above, we infer that if the best mode for the co-located macroblock in the base layer or the neighbouring macroblocks in the enhancement layer is MODE_SKIP, there is a high probability that the current macroblock in the enhancement layer is also MODE_SKIP, because of the strong dependency that exists.

2.3 DCT Coefficients and Picture Content

In a smooth region of an image that has been transformed using the discrete cosine transform (DCT), the DCT energy generally tends to be concentrated in the low frequency components. Whereas, in a block comprising high detail, the frequency domain energy is concentrated in the AC coefficients. On this basis, the sum of the AC coefficients can be chosen as a coarse measure of homogeneity.

For a 16×16 macroblock, the energy of the AC coefficients E_{AC} is calculated as

$$E_{AC} = \sum_{x=0}^{15}\sum_{y=0}^{15}\left(f\left(x,y\right)\right)^2 - \frac{1}{256}\left(\sum_{x=0}^{15}\sum_{y=0}^{15} f(x,y)\right)^2 \tag{4}$$

Yu *et al* [14] showed that for fast mode selection in H.264/AVC, when the total energy of the AC coefficients in the macroblock is greater than 92735, the macroblock is best categorised as containing high spatial detail, as shown in (5), and for this to be considered when choosing a reduced subset of modes for evaluation.

$$\text{homogeneity} = \begin{cases} high & \text{if } \mathrm{E_{AC}} > 92735 \\ low & \text{otherwise} \end{cases} \tag{5}$$

In the evaluation, an AC energy threshold of 92735 was chosen to categorise the homogeneity of the macroblock content.

2.4 Detection of Motion Activity

As mentioned in [10], not all lower layer up-sampling data is suitable for inter-layer prediction, especially for video sequences with slow motion and high spatial detail. Therefore it is important to identify the amount of motion as well as the spatial detail.

Motion vector difference (MVD) between key frames in each GOP is chosen as the assessment of motion. MVD is the difference between the actual motion vector and the predicted motion vector, defined as

$$|MVD| = |MV_{actual} - MV_p| \tag{6}$$

where MV_{actual} is the actual motion vector for the block and MV_p is the predicted motion vector.

Table 2 % Mode Correlation between Macroblock and its Neighbours.

Sequence	Qp				
	24	28	32	36	40
Bus	45.24	52.18	56.95	62.52	66.88
Foreman	45.00	55.24	61.93	69.26	77.53
Mobile	46.06	52.43	57.78	62.16	66.27
Mother-Daughter	71.73	82.05	87.07	92.86	96.16

Generally, sequences containing little motion tend to have small MVDs and vice versa. The MVD is easy to extract from the coded data, and this satisfies the overall objective of the research, to reduce computational complexity. In the evaluation, a MVD of 1.0 was chosen as the threshold.

3 Proposed Algorithm

The basic motivation is to reduce the number of mode candidates in the enhancement layer by exploiting the information in co-located macroblocks in the base layer and in neighbouring macroblocks. As the coded data of the base layer will be reused either directly or indirectly, its efficacy influences the performance of the encoder. The mode chosen for a macroblock in the base layer is estimated using an exhaustive search evaluation. As to the enhancement layer, our proposed algorithm is described as follows:

1. Check the mode of the co-located macroblock in the base layer. If it is intra coded, it means that a matching macroblock via inter prediction in the reference frames could not be found. Usually, such macroblocks contain fast changing or highly detailed information. For such macroblocks, we compare the RD cost of all the modes (including inter-layer prediction) and select the mode with minimum RD cost as the best mode for the current macroblock. Otherwise, proceed to step 2.
2. Check the co-located macroblock in the base layer and the neighbouring macroblocks. If at least one of these macroblocks is encoded with MODE_SKIP, we evaluate the RD cost of employing either MODE_SKIP or MODE_16×16. If mode MODE_SKIP has the least RD cost, it is chosen as the best mode for the current macroblock. Otherwise, proceed to step 3.
3. Measure the homogeneity of the macroblock content. In the case of low homogeneity, i.e. $E_{AC} \leq 92735$, large block partition sizes (MODE_16×16, MODE_16×8, MODE_8×16) require evaluation. Otherwise, proceed to step 4.
4. Observe the MVD which is easily extracted from the coded bit stream. If MVD<1, the macroblock contains little motion, and motion estimation can be performed with fewer candidates. Otherwise, more candidates are chosen corresponding to a larger search range.

4 Experimental Evaluation

The proposed algorithm was implemented using the JSVM 9.18 reference software [1]. Four standard video test sequences with diverse motion content were utilised with different Qp factors ranging from 24 to 40. The GOP size for the hierarchical B structure was set to 8 and over 90 frames were coded to generate a reliable result. We considered only the two-layer case. The same Qp was used for both base layer and enhancement layer. We chose computation Time Reduction (TR), bit-rate and PSNR as the performance measures. Table 3 shows the coding results of the standard JSVM 9.18 implementation and the proposed algorithm.

Table 3 shows that, in the case of the Bus sequence with fast motion activity and the Mobile sequence with high spatial detail, as the percentage of MODE_SKIP macroblocks in the base layer is small, the time reduction is lower than the other test sequences. Even so, the computation time is reduced by over 61%. For the Mother-Daughter sequence comprising little motion, an average time saving of 78% is achieved. In Table 4, it is apparent that the encoding time reduction increases as the value of Qp increases. The maximum time saved is 85% for the sequence containing slow motion. The proposed algorithm shows minimal degradation in coding efficiency with a bit-rate increment of no more than 1.6%, and a

Table 3 Performance when Encoding QCIF/CIF Sequences.

Sequence	Qp	JSVM		Proposed		TR
		BR(bits/s)	PSNR(dB)	BR(bits/s)	PSNR(dB)	
Bus	40	344.28	26.93	345.56	26.89	67.06
	36	565.78	29.44	570.09	29.39	64.36
	32	942.03	32.15	952.37	32.09	63.23
	28	1579.67	35.07	1593.86	34.98	61.28
	24	2581.37	38.00	2606.26	37.81	61.39
Foreman	40	157.93	30.28	157.76	30.25	73.19
	36	246.51	32.68	247.43	32.63	68.61
	32	390.35	34.96	392.54	34.91	65.07
	28	635.55	37.32	643.00	37.25	62.71
	24	1073.34	39.57	1090.13	39.47	60.53
Mobile	40	469.64	25.86	470.31	25.83	71.27
	36	752.69	28.44	754.84	28.41	69.85
	32	1314.27	31.25	1319.59	31.21	69.04
	28	2342.70	34.48	2359.49	34.42	68.58
	24	3940.10	37.71	3966.08	37.63	67.63
Mother-Daughter	40	66.83	32.24	66.66	32.22	84.59
	36	107.42	34.62	107.52	34.61	81.94
	32	175.36	37.08	175.43	37.07	78.42
	28	284.31	39.59	284.58	39.56	74.98
	24	463.07	41.87	463.36	41.83	69.45

Table 4 Overall Comparison of Proposed Algorithm and JSVM Implementation.

Sequence	Performance	Qp					Average
		40	36	32	28	24	
Bus	ΔPSNR(dB)	-0.04	-0.05	-0.06	-0.09	-0.19	-0.09
	ΔBR(%)	0.37	0.76	1.10	0.90	0.96	0.82
	TR(%)	67.06	64.36	63.23	61.28	61.39	63.46
Foreman	ΔPSNR(dB)	-0.03	-0.05	-0.05	-0.07	-0.1	-0.06
	ΔBR(%)	-0.11	0.37	0.56	1.17	1.56	0.71
	TR(%)	73.19	68.61	65.07	62.71	60.53	66.02
Mobile	ΔPSNR(dB)	-0.03	-0.03	-0.04	-0.06	-0.08	-0.05
	ΔBR(%)	0.14	0.29	0.40	0.72	0.66	0.44
	TR(%)	71.27	69.85	69.04	68.58	67.63	69.27
Mother-Daughter	ΔPSNR(dB)	-0.02	-0.01	-0.01	-0.03	-0.04	-0.02
	ΔBR(%)	-0.25	0.09	0.04	0.09	0.06	0.01
	TR(%)	84.59	81.94	78.42	74.98	69.45	77.88

decrease in PSNR of no more than 0.2%. Compared with Lee's algorithm [6], our proposed scheme achieves a greater time reduction for sequences with varying motion activity and spatial detail. The method also outperforms that of both Zhao [16] and Kim [5].

5 Conclusion

A fast hierarchical mode selection algorithm for the SVC extension of H.264/AVC has been described. The scheme reduces the number of mode candidates that need to be evaluated by exploiting the base layer information. Simulation results show that the algorithm achieves a reduction in encoding time of up to 85% with negligible reduction in coding efficiency and reconstructed video quality.

References

1 JSVM (Joint Scalable Video Model) reference software for SVC. Online. Available: CVS server garcon.ient.rwth-aachen.de.

2 H.264: Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 Advanced Video Coding, Mar. 2010.

3 Z. Y. Chen, J. W. Syu, and P. C. Chang. Fast inter-layer motion estimation algorithm on spatial scalability in H.264/AVC scalable extension. In *Proc. IEEE ICME*, pages 442–446, 2010.

4 G. Goh, J. Kang, M. Cho, and K. Chung. Fast mode decision for scalable video coding based on neighboring macroblock analysis. In *Proc. ACM Appl. Computing*, pages 1845–1846, 2009.

5 S. T. Kim, K. Reddy Konda, P. S. Mah, and S. J. Ko. Adaptive mode decision algorithm for inter layer coding in scalable video coding. In *IEEE Trans. Circuits Syst. Video Technol.*, pages 1297–1300, 2010.

6 B. Lee, M. Kim, S. Hahm, C. Park, and K. Park. A fast mode selection scheme in inter-layer prediction of H.264 scalable extension coding. In *Proc. IEEE BMSB*, pages 1–5, 2008.

7 H. Li, Z. G. Li, and C. Wen. Fast mode decision for coarse grain SNR scalable video coding. In *Proc. IEEE ICASSP*, volume 2, pages II–II, 2006.

8 X. Lu and G. R. Martin. Fast mode decision algorithm for the H.264/AVC scalable video coding extension. *IEEE Trans. Circuits Syst. Video Technol.*, 23(5):846–855, 2013.

9 H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable H.264/MPEG4-AVC extension. In *Proc. IEEE ICIP*, pages 161–164, 2006.

10 H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Trans. Circuits Syst. Video Technol.*, 17(9):1103–1120, 2007.

11 C. A. Segall and G. J. Sullivan. Spatial scalability within the H.264/AVC scalable video coding extension. *IEEE Trans. Circuits Syst. Video Technol.*, 17(9):1121–1135, 2007.

12 T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan. Rate-constrained coder control and comparison of video coding standards. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):688–703, 2003.

13 T. Wiegand, G. J. Sullivan, G. Bjϕntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):560–576, 2003.

14 A. C. W. Yu, G. R. Martin, and H. Park. Fast inter-mode selection in the H.264/AVC standard using a hierarchical decision process. *IEEE Trans. Circuits Syst. Video Technol.*, 18(2):186–195, 2008.

15 R. Zhang and M. L. Comer. Efficient inter-layer motion compensation for spatially scalable video coding. *IEEE Trans. Circuits Syst. Video Technol.*, 18(10):1325–1334, 2008.

16 T. Zhao, H. Wang, and S. Kwong. Fast inter-layer mode decision in scalable video coding. In *Proc. IEEE ICIP*, pages 4221–4224, 2010.

Improved Rate Control Algorithm for Scalable Video Coding*

Xin Lu and Graham R. Martin

Department of Computer Science
University of Warwick, Coventry
CV4 7AL, United Kingdom
{xin, grm}@dcs.warwick.ac.uk

Abstract

In the Scalable Video Coding (SVC) standard, a multi-layer based structure is utilised to support scalability. However in the latest Joint Scalable Video Model (JSVM) reference software, the rate control algorithm is implemented only in the base layer, and the enhancement layers are not equipped with a rate control scheme. In this work, a novel rate control algorithm is proposed for when inter-layer prediction is employed. Firstly, a Rate-Quantisation (R-Q) model, which considers the coding properties of different prediction modes, is described. Secondly, an improved Mean Absolute Difference (MAD) prediction model for the spatial enhancement layers is proposed, in which the encoding results from the base layer are used to assist the linear MAD prediction in the spatial/CGS enhancement layers. Simulation results show that, on average, rate control accuracy is maintained to within 0.07%. Compared with the default JVT-G012 rate control scheme employed in SVC, the proposed rate control algorithm achieves higher coding efficiency, namely an improvement of up to 0.26dB in PSNR and a saving of 4.66% in bitrate.

1998 ACM Subject Classification E.4 Coding and Information Theory

Keywords and phrases Inter-layer prediction, MAD prediction, Rate control, Scalable Video Coding (SVC), SVC extension of H.264/AVC.

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.73

1 Introduction

Scalable Video Coding (SVC), the scalable extension of the H.264/AVC standard, provides solutions for video applications with different network bandwidths, device capabilities and user demands. Bandwidth is a valuable resource, and often there are situations in which the bandwidth is insufficient or the network is unstable. Sometimes even the bitstream comprising the basic layer cannot be transmitted completely, resulting in frame skipping. In these cases, effective rate control is essential. In real-time applications, rate control enables the output bit rate to adjust quickly depending on the available channel bandwidth. With a proper control scheme, "overflow" and "underflow" of the buffer are prevented, which means that frame skipping and wastage of channel resources can be avoided. Thus, rate control extends the scalability of SVC. Furthermore, rate control appropriately allocates the available bits according to the complexity of the image content, so that the quality of the video is maximised.

Several rate control algorithms have been proposed for non-scalable video coders, such as the Test Model 5 (TM5) [10] for MPEG-2, Test Model Near-term 8 (TMN8) [5] for

* A longer version of this paper appeared in the proceedings of IEEE MMSP'13 [9].

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 73–80

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

H.263, Verification Model 8 (VM8) [11] for MPEG4 and JVT-G012 [6] for H.264/AVC. The JVT-G012 rate control algorithm is only implemented in the base layer of the latest JSVM reference software, and it does not support the enhancement layers which provide the scalability functions. Rate control algorithms that address the properties of the enhancement layers in SVC need to be developed. Several rate control algorithms have been suggested for SVC [16, 8, 4, 7]. They consider either precise target bit allocation or the optimisation of the rate quantisation model. Xu *et al.* proposed a rate control algorithm for spatial and Coarse Gain SNR (CGS) scalable coding in SVC [16]. This method employs the improved TMN8 model for quantisation parameter (Qp) estimation based on the mode analysis of I, P, and B frames. In [8], Liu *et al.* proposed that the MAD can be predicted from either the previous frame in the same layer or the corresponding frame in the base layer through a switching law. Hu *et al.* [4] proposed a frame level rate control algorithm for temporal scalability of scalable video coding by developing a set of weighting factors for bit allocation. In [7], Liu *et al.* proposed a bit allocation algorithm for SVC when the inter-layer dependency is taken into consideration.

SVC employs so called inter-layer prediction to reduce the redundancies between layers. However, the effects of inter-layer prediction are not taken into consideration in the rate control scheme of the JSVM. The rate control strategies assume that the statistical property of a video source is fixed [15], and then they derive a precise rate distortion model [12]. From observation and analysis, macroblocks coded using inter-layer prediction and those coded by intra-layer prediction (inter-frame prediction and intra-frame prediction) have dissimilar statistical properties. Furthermore, some encoding results of the base layer can be used to inform the encoding of the enhancement layers, thus benefiting from the bottom-up coding structure of SVC. These observations motivate us to propose a rate control scheme with a precise Rate-Quantisation (R-Q) model and optimised MAD prediction for the spatial enhancement layers.

The remainder of this paper is organised as follows. Section 2 discusses the formulation of the proposed rate distortion model, and the proposed MAD prediction scheme is presented in Section 3. Extensive experimental results are presented in Section 4 and conclusions are given in Section 5.

2 Rate-Distortion Model for Spatial Enhancement Layer

With the hypothesis that the residual coefficients obey a Laplacian distribution, the classic quadratic rate-distortion (R-D) model is described as follows [3],

$$R_{txt} = \frac{X_1 \times MAD_{pred}}{Q_{step}^2} + \frac{X_2 \times MAD_{pred}}{Q_{step}} \tag{1}$$

where R_{txt} is the target number of bits assigned to code the texture information of a basic unit; MAD_{pred} indicates the mean absolute difference of the residual component; Q_{step} is the quantisation step size to be calculated and X_1 and X_2 are model coefficients. X_1 and X_2 are updated using a linear regression method after the coding of each basic unit [3].

In SVC, the inter-layer prediction tools are employed to reduce the redundancies between layers. However, inter-layer prediction is not efficient for coding sequences containing homogeneous texture or slow motion, since the high frequency components in the enhancement layer cannot be reconstructed well by upsampling the information of the base layer. Macroblocks with little detail and slow motion are more likely to be best matched with a block by inter-frame prediction in the same layer. Consequently, temporal prediction (inter-frame

Table 1 Relationship between average number of texture bits per coded unit and Qp for both inter-layer predicted macroblocks and intra-layer predicted macroblocks.

Sequence	Prediction mode	Qp				
		28	32	36	40	44
Bus	Intra-layer	98.98	47.91	18.53	6.26	1.66
	Inter-layer	98.68	52.22	28.05	15.28	7.94
Football	Intra-layer	77.89	43.92	22.07	9.41	3.79
	Inter-layer	100.3	56.63	32.12	20.31	11.76
Foreman	Intra-layer	18.61	6.52	2.40	0.86	0.39
	Inter-layer	28.81	14.27	7.59	4.72	3.20
Mobile	Intra-layer	185.2	77.89	22.57	6.76	2.23
	Inter-layer	162.4	96.88	46.41	22.33	12.79

prediction) is more efficient than inter-layer prediction, especially for sequences with slow motion. In contrast, inter-layer prediction performs well for fast moving sequences. However, macroblocks with fast movement usually need more bits to encode. So we observe that the average number of texture bits for inter-layer predicted macroblocks in the enhancement layers is significantly greater than for macroblocks which have been coded using temporal and spatial prediction within the same enhancement layer.

The significant difference in the number of texture bits generated for inter-layer predicted macroblocks and intra-layer predicted macroblocks leads to serious prediction errors in the rate-distortion model coefficients. To illustrate the mathematical relationship between Qp and the texture bits for both inter-layer predicted macroblocks and intra-layer predicted macroblocks, and to justify our proposed algorithm, we analysed the simulation results from processing four sequences with different degrees of activity and detail. The statistics were collected from the first 150 frames of each video sequence. QCIF sequences were used for the base layer and CIF were used for the enhancement layer.

Table (1) shows the average number of texture bits for inter-layer predicted macroblocks in the enhancement layer and those for intra-layer predicted macroblocks in the same enhancement layer, under a variety of Qp values. It is observed that the average number of bits used for encoding inter-layer predicted macroblocks is significantly different from that required for intra-layer predicted macroblocks. In general, as Qp increases, significantly more bits are consumed by inter-layer prediction coding than by intra-layer prediction.

The model relationship between Qp and Q_{step} is

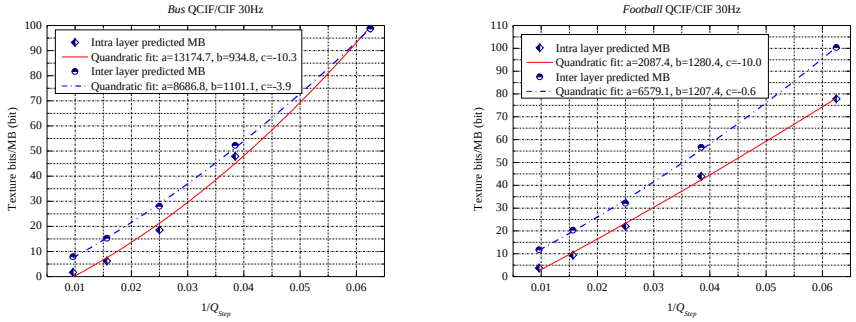
$$Qp = 2^{\frac{Q_{step}}{6}} \zeta(Q_{step}\%6) \tag{2}$$

where $\zeta(0)$=0.675; $\zeta(1)$=0.6875; $\zeta(2)$=0.8125; $\zeta(3)$=0.875; $\zeta(4)$=1.0; $\zeta(5)$=1.125 [13].

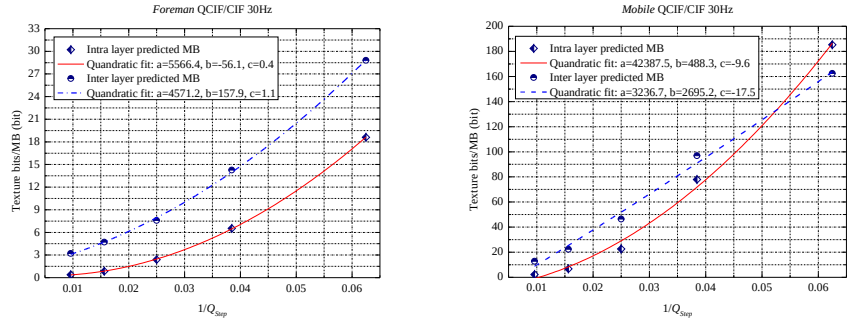
Figure (1) illustrates the relationship between Q_{step} values and the obtained number of texture bits R_{txt} for both inter-layer predicted macroblocks and intra-layer predicted macroblocks. The quadratic curves fitting the measured data are also presented. It can be seen that the measured data can be represented by quadratic functions very well.

$$R_{txt} = \frac{a}{Q_{step}^2} + \frac{b}{Q_{step}} + c \tag{3}$$

where $c \approx 0$. The coefficients of the quadratic model are obtained by finding the minimal fitting error. Although the observed R_{txt} - Q_{step} relationship can be represented by quadratic models, the model coefficients are significantly different. From these observations, it is concluded that for optimised rate control within the enhancement layers, separate, and



(a) *Bus.* **(b)** *Football.*



(c) *Foreman.* **(d)** *Mobile.*

Figure 1 Relationship between average number of texture bits and Q_{step} for both inter-layer coding and non-inter-layer coding. Points are actual data; curves are fitted to the data.

sufficiently accurate, models must be used for inter-layer prediction and intra-layer prediction. Consequently, Q_{step}^{inter} and Q_{step}^{intra} can be modelled accurately by quadratic functions with their respective model coefficients.

$$\begin{aligned} R_{txt} &= \frac{X_1^{inter} \times MAD_{pred}}{\left(Q_{step}^{inter}\right)^2} + \frac{X_2^{inter} \times MAD_{pred}}{Q_{step}^{inter}} \\ R_{txt} &= \frac{X_1^{intra} \times MAD_{pred}}{\left(Q_{step}^{intra}\right)^2} + \frac{X_2^{intra} \times MAD_{pred}}{Q_{step}^{intra}} \end{aligned} \tag{4}$$

where R_{txt} is the target number of texture bits for the current basic unit; MAD_{pred} is the MAD predicted from the previous coding results, Q_{step}^{inter} and Q_{step}^{intra} are the desired quantisation step sizes for inter-layer prediction and intra-layer prediction respectively, X_1^{inter} and X_2^{inter}, X_1^{intra} and X_2^{intra}, are the model coefficients for inter-layer prediction and intra-layer prediction, each updated after the coding of a basic unit using inter-layer prediction and intra-layer prediction respectively.

3 Optimisation of MAD Prediction for Spatial Enhancement Layer

From equations (4) it can be seen that the quantisation step sizes Q_{step}^{inter} and Q_{step}^{intra} depend on the model coefficients, the target number of bits R_{txt} for the current basic unit, and the predicted MAD value of the current basic unit. However, the MAD value is unknown before

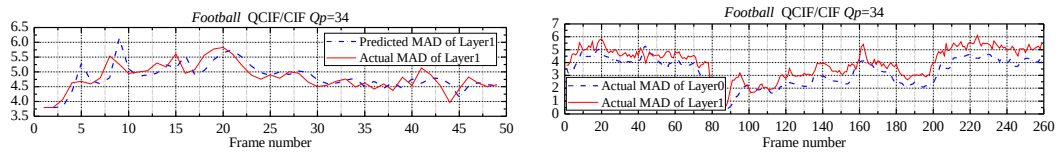


Figure 2 The MAD relationships (part of *Football* sequence) (a) Predicted and actual MAD values; (b) Actual MAD values of base layer and spatial enhancement layer.

rate-distortion optimisation (RDO). The MAD value can only be obtained after coding the current basic unit using the quantisation step size, but the model needs the MAD value to calculate the quantisation step size. This is a "chicken and egg situation". The JVT-G012 rate control algorithm overcomes this problem by using the MAD value of the basic unit in the same position of the previous frame to predict the MAD value of current basic unit, thus permitting the quantisation step size to be calculated. A linear MAD model is adopted here as [6]:

$$\mathrm{MAD_j} = \mathrm{a_1} \times \mathrm{MAD_{j-1}} + \mathrm{a_2} \tag{5}$$

where $\mathrm{a_1}$ and $\mathrm{a_2}$ are model coefficients, updated after the coding of each frame. $\mathrm{MAD_j}$ denotes the predicted MAD of the current basic unit and $\mathrm{MAD_{j-1}}$ denotes the actual MAD of the basic unit in the corresponding position of the previous frame.

In the quadratic R-D model, MAD prediction is very important as it directly affects the allocation of bits. As the prediction is not always accurate, there is always some small error in bit allocation, and this cannot be avoided in the JVT-G012 algorithm. As shown in Figure (2)(a), if the MAD fluctuates due to fast motion or scene changes in the video sequence, the linear model performs poorly, and there is always a delay. In the example, this phenomenon is particularly obvious at frames 9, 41 and 44.

In the SVC encoding process, for each frame, the base layer is encoded first, prior to the enhancement layers. Furthermore, the content of the base layer and enhancement layers are highly correlated. As shown in Figure (2)(b), even though the MAD values of the two layers are not the same, they have a similar tendency in the presence of abrupt changes. This leads to the idea that some encoding results of the base layer can be used to inform the coding of the enhancement layer(s), thus benefitting from the bottom-up coding structure of the standard. Therefore, a new MAD prediction model for the spatial enhancement layer using the encoding results from the base layer as a factor in the MAD prediction procedure is proposed. The new prediction model is defined as:

$$\mathrm{E\_MAD_j} = \mathrm{a_1} \times \mathrm{E\_MAD_{j-1}} + \mathrm{a_2} + \mathrm{a_3} \times \Gamma_\mathrm{j} \tag{6}$$

As for equation (5), $\mathrm{a_1}$ and $\mathrm{a_2}$ are model coefficients updated after the coding of each frame. The prefix E_ indicates the enhancement layer. Note that the model is a general model and can be used at the basic coding unit (e.g. macroblock) level or at frame level. Therefore, $\mathrm{E\_MAD_j}$ may refer to the predicted MAD value of the $\mathrm{j^{th}}$ frame or that of one basic unit in the $\mathrm{j^{th}}$ frame. Similarly, $\mathrm{E\_MAD_{j-1}}$ may refer to the actual MAD value of the previous frame or that of one basic unit in the same position of the previous frame. Γ_j refers to the difference between the actual and predicted MAD of the base layer of the $\mathrm{j^{th}}$ frame in the base layer, and is defined as

$$\Gamma_\mathrm{j} = \mathrm{B\_MAD_{actual,j}} - \mathrm{B\_MAD_{predicted,j}} \tag{7}$$

Table 2 Comparison of rate control accuracy.

Sequence	Target bitrate (kbps)	FixedQp		JVT-G012		Proposed	
		BR (kbps)	Mism. (%)	BR (kbps)	Mism. (%)	BR (kbps)	Mism. (%)
Bus	384	391.5	1.94	385.0	0.26	384.4	0.10
	512	522.1	1.96	513.3	0.25	512.0	0.00
	768	745.6	2.91	769.1	0.14	768.6	0.08
	1280	1303.9	1.87	1282.1	0.16	1280.6	0.05
Football	768	743.2	3.23	768.5	0.07	768.0	0.00
	1024	1042.3	1.79	1024.5	0.05	1024.6	0.06
	1536	1519.7	1.06	1538.3	0.15	1535.5	0.03
	2560	2513.3	1.82	2560.1	0.00	2559.2	0.03
Foreman	192	191.8	0.12	192.9	0.47	192.3	0.16
	256	257.1	0.42	256.7	0.27	256.3	0.12
	384	389.2	1.35	385.0	0.26	384.3	0.08
	640	637.7	0.36	641.3	0.20	639.8	0.03
Mobile	256	251.3	1.83	256.1	0.04	256.6	0.23
	384	370.3	3.57	384.7	0.18	384.4	0.10
	512	522.0	1.96	512.8	0.16	512.1	0.02
	768	777.4	1.22	769.0	0.13	767.6	0.05
Average			**1.71**		**0.17**		**0.07**

where the prefix B_ indicates the base layer. a_3 is the Γ_j weighting factor and typically is assigned a value of 0.1. The value of a_3 is determined from consideration of a very large number of training samples and many different types of picture content. Consequently it is reliable and widely-applicable. It may be possible to define an adaptive threshold that is even more accurate, and this may be pursued in the future.

With the above model, the prediction errors from the base layer are used to assist estimation of the MAD in the enhancement layers. When encoding the enhancement layers, the encoder is made aware of the abrupt changes of MAD in advance and promptly adjusts the MAD prediction to reduce the prediction errors. In this way, the bits are allocated more appropriately and not only is there an improvement in the rate control accuracy, but also an increase in the quality of the reconstructed video.

4 Experimental Results

The proposed algorithm was incorporated in the SVC reference software JSVM9.19.14 [1]. In order to validate the effectiveness of the proposed algorithm, video sequences with different degrees of motion activity and picture detail were coded, and the results compared with the JVT-G012 algorithm. The first 150 frames of each video sequence were coded to generate a reliable result. Two spatial layers are evaluated and the proposed algorithm is applied to the enhancement layer. Adaptive inter-layer prediction is enabled for the enhancement layer. As our algorithm attempts to optimise the R-Q model and MAD prediction model, which are only involved in P frames, the GOP structure is set to IPPP. In this work, a macroblock is chosen as the basic unit for rate control. To compare the results with the JVT-G012 algorithm, the initial Qp value is set to 32 for both schemes. Other parameters are set to the default values of the reference software.

The bit-rate mismatch (%Mism.) and rate distortion performance in terms of , BDBR (%), BDPSNR (dB), ΔBR (%), and ΔPSNR (dB) [2] were measured against the JVT-G012 scheme to evaluate the coding performance of the proposed rate control algorithm. ΔPSNR(dB) is computed according to $\Delta\text{PSNR} = \text{PSNR}_{\text{Proposed}} - \text{PSNR}_{\text{G012}}$, where $\text{PSNR}_{\text{Proposed}}$ and $\text{PSNR}_{\text{G012}}$ denote the PSNR resulting from the proposed algorithm and JVT-G012, and

Table 3 Comparison of rate distortion performance.

Sequence	JVT-G012		Proposed		ΔBR	ΔPSNR	BDBR	BDPSNR
	BR (kbps)	PSNR (dB)	BR (kbps)	PSNR (dB)	(%)	(dB)	(%)	(dB)
Bus	385.0	28.00	384.4	28.13	-0.16	0.13	-3.55	0.17
	513.3	29.28	512.0	29.43	-0.25	0.15		
	769.1	31.10	768.6	31.28	-0.07	0.18		
	1282.1	33.67	1280.6	33.82	-0.12	0.15		
Football	768.5	32.94	768.0	33.11	-0.07	0.17	-4.66	0.26
	1024.5	34.29	1024.6	34.59	0.01	0.30		
	1538.3	36.53	1535.5	36.78	-0.18	0.25		
	2560.1	39.39	2559.2	39.66	-0.04	0.27		
Foreman	192.9	32.85	192.3	32.97	-0.31	0.12	-2.68	0.11
	256.7	34.06	256.3	34.17	-0.16	0.11		
	385.0	35.69	384.3	35.78	-0.18	0.09		
	641.3	37.64	639.8	37.77	-0.23	0.13		
Mobile	256.1	23.98	256.6	24.07	0.20	0.09	-4.01	0.16
	384.7	25.61	384.4	25.71	-0.08	0.10		
	512.8	26.64	512.1	26.83	-0.14	0.19		
	769.0	28.08	767.6	28.38	-0.18	0.30		
Average					**-0.12**	**0.17**	**-3.73**	**0.18**

ΔBR(%) is computed as $\Delta BR = (BR_{Proposed} - BR_{G012})/BR_{G012} \times 100\%$, where $BR_{Proposed}$ and BR_{G012} denote the bit-rate resulting from the proposed algorithm and JVT-G012, respectively.

The rate control accuracy for four target bit rates is summarised in Table (2). All the test sequences and bitrates used in the experiments are those recommended by the JVT in document JVT-Q205 [14]. It can be seen that both the proposed algorithm and the JVT-G012 scheme work well at various target bit rates. Although both methods produce the target bit rates, the accuracy of the proposed algorithm is better than JVT-G012 in most cases. This is because the proposed optimised MAD prediction model results in a smaller prediction error when fast motion occurs. Most of the mismatch errors are less than 0.1% and the maximum error is 0.23%. The overall average absolute mismatch error is 0.07%. Consequently, it can be considered that bit rate is precisely controlled using the proposed algorithm.

The proposed rate control mechanism also achieves better rate-distortion performance for the enhancement layers than the JVT-G012 scheme. The comparative performance results are shown in Table (3). The results show that 1) given the same bit rate, the proposed algorithm increases the average PSNR by up to 0.26dB, and 2) given the same video quality (PSNR), the proposed algorithm produces a saving in average bit rate of up to 4.66%, compared to the JVT-G012 algorithm. In general, the PSNR of each of the four sequences is increased at all ranges of target bit rate. The maximum coding gain is 0.30dB. Therefore, the proposed algorithm improves the coding efficiency compared with the JVT-G012 rate control algorithm, and this is true regardless of target bit rate.

In order to test the robustness of the proposed algorithm, it was applied to video sequences of larger spatial resolution and with multiple enhancement layers. The experimental results show that the proposed algorithm consistently achieves a significant improvement in RD performance compared with that of JVT-G012. Due to limitations on space, the detailed results are presented in the longer version of this paper.

5 Conclusions

A rate control scheme for the spatial enhancement layer(s) in SVC has been described. The scheme introduces a separate rate-quantisation (R-Q) model for inter-layer prediction coding

in the enhancement layer. An improved MAD prediction model is also proposed, where the MAD from previous temporal frames and previous spatial frames are considered together. In applying each of the above techniques, both the target bit rate mismatch is reduced and the coding efficiency is significantly improved. Simulation results show that the proposed method achieves better rate control accuracy than the JVT-G012 scheme, the average rate control mismatch error being 0.07%. Furthermore, the proposed algorithm attains higher coding efficiency than the JVT-G012 rate control algorithm. The improvement, averaged over the different types of video sequences coded, is an increase in PSNR of 0.18dB or a saving in bit rate of 3.73%.

References

1 JSVM (Joint Scalable Video Model) reference software for SVC. Online. Available: CVS server garcon.ient.rwth-aachen.de.

2 G. Bjϕntegaard. Calculation of average PSNR differences between RD-curves. *VCEG-M33*, Apr. 2001.

3 T. Chiang and Y. Zhang. A new rate control scheme using quadratic rate distortion model. *IEEE Trans. Circuits Syst. Video Technol.*, 7(1):246–250, 1997.

4 S. Hu, H. Wang, S. Kwong, T. Zhao, and C.-C. J. Kuo. Rate control optimization for temporal-layer scalable video coding. *IEEE Trans. Circuits Syst. Video Technol.*, 21(8):1152–1162, 2011.

5 ITU-T Study Group 16. Video Codec Test Model, Near-Term, Version 8 (TMN8). *ITU-T/SG16/VCEG/Q15 A59*, Jun. 1997.

6 Z. G. Li, W. Gao, F. Pan, S. W. Ma, K. P. Lim, G. N. Feng, X. Lin, S. Rahardja, H. Q. Lu, and Y. Lu. Adaptive rate control for H.264. *J. Vis. Commun. Image Represent.*, 17(2):376–406, 2006.

7 J. Liu, Y. Cho, Z. Guo, and C.-C. J. Kuo. Bit allocation for spatial scalability coding of H.264/SVC with dependent rate-distortion analysis. *IEEE Trans. Circuits Syst. Video Technol.*, 20(7):967–981, 2010.

8 Y. Liu, Z. Li, and Y. Soh. Rate control of H.264/AVC scalable extension. *IEEE Trans. Circuits Syst. Video Technol.*, 18(1):116–121, 2008.

9 X. Lu and G. R. Martin. Rate control for scalable video coding with rate-distortion analysis of prediction modes. In *Proc. IEEE MMSP*, pages 289–294, 2013.

10 MPEG-2 Video Test Model Editing Committee. MPEG-2 Video Test Model 5 (TM5). *ISO/IEC JTC1/SC29/WG11 N0400*, Apr. 1993.

11 MPEG Video Group. MPEG-4 Video Verification Model Version 8 (VM8). *ISO/IEC JTC1/SC29/WG11 N1796*, Jul. 1997.

12 J. Ribas-Corbera and S. Lei. Rate control in DCT video coding for low-delay communications. *IEEE Trans. Circuits Syst. Video Technol.*, 9(1):172–185, 1999.

13 I. E. Richardson. *H.264 and MPEG-4 video compression: video coding for next-generation multimedia*. Wiley, 2003.

14 M. Wien and H. Schwarz. Testing conditions for SVC coding efficiency and JSVM performance evaluation. *JVT-Q205*, Jul. 2005.

15 J. Xie and L. Chia. Study on the distribution of DCT residues and its application to R-D analysis of video coding. *J. Vis. Commun. Image Represent.*, 19(7):411–425, Oct. 2008.

16 L. Xu, W. Gao, X. Ji, D. Zhao, and S. Ma. Rate control for spatial scalable coding in SVC. In *Proc. PCS*, 2007.

An Optimal Real-time Pricing Algorithm for the Smart Grid: A Bi-level Programming Approach*

Fan-Lin Meng and Xiao-Jun Zeng

School of Computer Science, University of Manchester
Manchester, United Kingdom
mengf@cs.man.ac.uk, x.zeng@manchester.ac.uk

Abstract

This paper proposes an improved approach to our previous work [11]. [11] uses Stackelberg game to model the interactions between electricity retailer and its customers and genetic algorithms are used to obtain the Stackelberg Equilibrium (SE). In this paper, we propose a bi-level programming model by considering benefits of the electricity retailer (utility company) and its customer. In the upper level model, the electricity retailer determines the real-time retail prices with the aim to maximize its profit. The customer reacts to the prices announced by the retailer aiming to minimize their electricity bills in the lower level model. In order to make it more tractable, we convert the hierarchical bi-level programming problem into one single level problem by replacing the lower lever's problem with his Karush–Kuhn–Tucker (KKT) conditions. A branch and bound algorithm is chosen to solve the resulting single level problem. Experimental results show that both the bi-level programming model and the solution method are feasible. Compared with the genetic algorithm approach proposed in work [11], the branch and bound algorithm in this paper is more efficient in finding the optimal solution.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Real-time Pricing, Demand Response, Smart Gird, Bi-level Programming, Branch and Bound Algorithm

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.81

1 Introduction

The traditional electricity grid is facing many existing and potential problems with the increased demand from customers in recent years, and the reliability of the grid has been put in danger. In addition, the average household electricity load has the potential to double with the deployment of plug-in hybrid electric vehicles (PHEVs), which will further endanger the existing grid.

Instead of building more power plants to meet the peak demand of customers, demand response is a better choice for solving the above problems, especially with the development of the smart grid.

Real-time pricing (RTP) is one of the most important DR strategies, where the prices announced by retailers change typically hourly to reflect variations of the price in the wholesale market over time. Generally, customers are notified of RTP prices the day before or a few hours before the delivery time. One of the most typical types of RTP is day-ahead RTP, in which customers receive the prices for the next 24 hours [6].

* Parts of this paper appeared in the proceedings of UKCI 2012 [11].

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andew V. Jones, Nicholas Ng; pp. 81–88

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There exists much literature on RTP. However, the results and analysis in this paper differ from the related work in several aspects:

In the work of [16], they analytically models the customers' preferences and customers' electricity consumption patterns in form of utility functions and it shows that the proposed algorithm can benefit both customers and energy providers. However, no explicit form of the customers' utility functions is given. [8] and [18] further develop the work of [16]. Both works use the same concept of utility functions to model the satisfaction of customers as [16], but similarly no explicit form of the utility functions are given. As a result, the approach is unable to help the customers to find the best scheme to minimize their bills. To overcome this weakness, the approach given in this paper aims to provide the best solution for customers to achieve the minimal bills.

Since the RTP design needs the participation of electricity retailer and its customers and the decision makings are sequential, i.e., the electricity retailer announces the prices first, then its customers react to the prices by shifting the energy use. The interactions between electricity retailer and its customers can be represented as a leader-follower Stackelberg game, and thus can be modelled using a bi-level programming model. In fact, the bi-level programming problem is a static Stackelberg game where two players try to maximize their individual objective functions [1].

Due to the hierarchical structure of the Stackelberg game or bi-level programming model, many real-world problems with two decision levels can be modelled using the bi-level programming approach. Price setting problems are two decision levels problems and have been studied for several years using bi-level programming approach [14, 15, 7]. [2] proposes a decision-making scheme for electricity retailers based on Stackelberg game. They model the customers' preference and satisfaction as utility functions. [4] presents an optimal demand response scheduling with Stackelberg game approach. Similar to [2], they model the customers' behaviour patterns as utility functions. However, no explicit form of utility functions are given. The difference between our work and [2] and [4] lies in that we model the follower level problem (lower level problem) with appliance-level details, which is more practical and thus more difficult to solve.

The main focus of this paper is to propose a decision making scheme based on bi-level programming model for the electricity retailer and its customer by considering the benefits of both participants and give efficient solutions to the proposed model.

The rest of this paper is organized as follows. The background of bi-level programming model is introduced in Section 2. In Section 3, the system model and the solution method are given. Experimental results are presented in Section 4. The paper is concluded in Section 5.

2 Background of Bi-level Programming Model

Decision making problems in decentralized organizations are often modelled as Stackelberg games, and they are formulated as bi-level mathematical programming problems. The major feature of a bi-level programming problem is that it includes two optimization problems within a single instance. The lower level executes its own optimal policy after decisions are made at the upper level [10].

The general formulation of a bi-level programming problem can be represented as follows:

$$\begin{aligned} (Upper \quad Level) \quad & \min_{x} F(x, y) \\ s.t. \quad & G(x, y) \leq 0 \end{aligned}$$

where $y = y(x)$ is implicitly defined by:

$$\begin{aligned} (Lower \quad Level) \quad & \min_{y} f(x, y) \\ s.t. \quad & g(x, y) \leq 0 \end{aligned}$$

where F is the objective function of the upper level problem; f is the objective function of the lower level problem; G is the constraint set of the upper level decision vector; g is the constraint set of the lower level decision vector; x is the decision vector of the upper level; and y is the decision vector of the lower level. $y = y(x)$ is called reaction function. To solve the bi-level programming model, one needs to obtain the reaction function by solving the lower-level problem and replace the variable y in the upper level problem with the reaction function [17]. However, for many real applications, the reaction function $y = y(x)$ is not able to be explicitly represented. Thus, the problems can not be handled in the above way and become more difficult to solve.

Even though the simple linear bi-level programming problems are proven to be NP-hard, there are many methods arising in the last thirty years for solving the bi-level programming problems, such as the extreme-point approach for the linear bi-level programming, branch and bound method, descent methods, penalty function methods and trust region methods [5]. In this paper, the branch and bound method is chosen to solve our proposed bi-level programming problems as this algorithm was proved to be able to obtain the global optimal solutions [1].

3 System Model and Solution Method

In this section, we provide a mathematical representation of the considered decision making problem. Firstly, our focus is to formulate the electricity consumption scheduling problem in response to the real-time pricing in each household as an optimization problem that aims to minimize the payment bills. Secondly, we model the profit optimization problem for the retailer who will offer the 24 hours real-time prices to the customer.

We define $P = [p^1, p^2, ..., p^h, ...p^H]$ as the leader's strategy space, where p^h represents the electricity price at hour h and H represents the scheduling time window. We assume that $p^{min} \leq p^h \leq p^{max}$, where p^{min} represents the minimum price that the retailer (utility company) can offer to the customer and p^{max} represents the maximum price that the retailer can offer. It is also reasonable to assume that the price that the retailers can offer is greater than the wholesale price of each hour. The prices of p^{min} and p^{max} are usually designed based on history data and conditions of the wholesale price. However, since most of the retail markets now are regulated, there exists a price cap for the retail price and p^{max} should be less than the price cap. For the following part of this paper, we set $\mathcal{H} \triangleq \{1, 2, ..., H\}$. Usually, $H = 24$. We define the set of appliances in the customer's household S. In this paper, we only consider the one-leader, one-follower case, i.e., in our model, only one electricity retailer and one customer are considered.

3.1 Lower-level Model Problem

This model improves that of [13]. In their work, a upper limit for hourly electricity usage is set for each household, but we do not have such constraints for the optimization problem at customer's side as there is no such usage limits in practice. Instead, we consider the total

upper limit of hourly usage in the optimization problem at retailers' side. This is to represent the maximum load capacity of power networks. Therefore, we can actually control the hourly use of electricity of each household by properly determining the retail price, which is more practical from an application point of view.

For each appliance $s \in S$, we define an electricity consumption scheduling vector:

$$\mathrm{e}_s = [e_s^1, ..., e_s^h, ..., e_s^H] \tag{1}$$

where H is the scheduling window. For each hour $h \in \mathcal{H} \triangleq \{1, 2, ..., H\}$, $e_s^h \geq 0$ represents the customer's electricity consumption of appliance s at time h.

It is reasonable to assume that the energy consumption of each appliance s during a typical day is maintained at the same level and the total electricity consumed by appliance s in a typical day is defined as E_s. Moreover, the customer needs to set a valid scheduling window $\mathcal{H}_s \triangleq \{\alpha_s, ..., \beta_s\}$ by specifying the beginning operation time $\alpha_s \in \mathcal{H}$ and the end operation time $\beta_s \in \mathcal{H}$ of appliance s. Based on the above analysis, we have

$$\sum_{h=\alpha_s}^{\beta_s} e_s^h = E_s \tag{2}$$

and

$$e_s^h = 0, \forall h \in \mathcal{H} \backslash \mathcal{H}_s \tag{3}$$

After defining the minimum power level γ_s^{min} and the maximum power level γ_s^{max} for each appliance $s \in S$, we have

$$\gamma_s^{min} \leq e_s^h \leq \gamma_s^{max}, \forall h \in \mathcal{H}_s. \tag{4}$$

Then, the payment bill optimization problem for the customer can be modelled as follows:

$$\begin{array}{l} \min\limits_{\boldsymbol{e_s^h}} \sum_{h=1}^{H} p^h \times (\sum_{s \in S} e_s^h) \\ s.t. \\ \sum_{h=\alpha_s}^{\beta_s} e_s^h = E_s, \\ e_s^h = 0, \forall h \in \mathcal{H} \backslash \mathcal{H}_s, \\ \gamma_s^{min} \leq e_s^h \leq \gamma_s^{max}, \forall h \in \mathcal{H}_s. \end{array} \tag{5}$$

3.2 Upper-level Model Problem

In this section, we model the profit of the retailer by using the revenue subtracting the energy cost imposed on the retailer. We will discuss about the energy cost model first, and then a profit maximization model will be proposed.

In the practical application scenario, to determine the retail price, we need to consider many factors such as running cost of the retailers including the payments incurred in the wholesale market and so on. For simplicity, we define a cost function $C_h(L_h)$ indicating the cost of providing electricity by the retailers at each hour $h \in \mathcal{H}$, where L_h represents the amount of power provided to the customer at each hour of the day. We assume that the cost function $C_h(L_h)$ is increasing in L_h for each h [12, 8, 3]. In view of this, we design the cost function as follows [12].

$$C_h(L_h) = a_h L_h + b_h \tag{6}$$

where $a_h > 0$ and $b_h \geq 0$ at each hour $h \in \mathcal{H}$.

For each hour $h \in \mathcal{H}$, by defining the minimum price that the retailer (utility company) can offer p^{min} and the maximum price p^{max}, we have $p^{min} \leq p^h \leq p^{max}$. Note that there is usually a maximum load capacity, denoted as $E_h^{\max}$, of power networks at each hour. Thus, we have following constraints:

$$\sum_{s \in S} e_s^h \leq E_h^{\max}, \forall h \in \mathcal{H} \tag{7}$$

Then the profit maximization problem can be modelled as (8).

$$\begin{aligned} &\max_{\boldsymbol{p^h}} \{ \sum_{h \in \mathcal{H}} p^h \times \sum_{s \in S} e_s^h - \sum_{h \in \mathcal{H}} C_h(\sum_{s \in S} e_s^h) \} \\ &s.t. \\ &p^{min} \leq p^h \leq p^{max} \\ &\sum_{s \in S} e_s^h \leq E_h^{\max}, \forall h \in \mathcal{H} \end{aligned} \tag{8}$$

3.3 Solution Method

Instead to solve the bi-level problem in its hierarchical form (Eqs.(8) and (5)), we convert it into a standard mathematical program by replacing the follower's problem (lower level problem, Eq.(5)) with his Karush–Kuhn–Tucker (KKT) conditions. Then a branch and bound algorithm is chosen to solve the resulting non-linear programming problem [1]. We adopt the YALMIP solver, which is based on the above mentioned algorithm and implemented in Matlab, to solve our bi-level programming mode [9].

4 Experimental Results

We simulate a simple one energy retailer (utility company), one customer case. It is assumed that the customer has 4 appliances: dish washer, washing machine, clothes dryer and PHEV. Note that the scheduling horizon is from 8AM to 8AM (the next day).

For the cost of the energy provided to the customer by utility company, we model this as a cost function. We choose a simple linear cost function: $C_h(L_h) = a_h L_h + b_h$, where L_h represents the amount of power provided to the customer at each hour of the day. For simplicity we assume that $b_h = 0$ for all $h \in \mathcal{H}$. Also, we have $a_h = 5.5$ cents during the day, i.e., from 8AM to 12AM and $a_h = 4.0$ cents at night hours, i.e., from 13AM to 8AM (the next day). Finally, the parameter settings of these home appliances can be found in Table 1.

Getting ideas from the time-of-use pricing (ToU), we divide the 24 hours prices into three levels, i.e., peak hours(5PM-12AM), mid-peak hours(8AM-5PM) and off-peak hours(12AM-8AM). For peak hours, the prices range from 12 cents to 14 cents. Similarly, the prices range

Table 1 Home Appliances' Parameter Settings.

Appliance Name	E_s	H_s	γ_s^{min}	γ_s^{max}
Dish washer	1.8kwh	8PM-6AM	0.1kwh	1.0kwh
Washing machine	1.94kwh	8AM-8PM	0.1kwh	1.0kwh
Clothes dryer	3.4kwh	7PM-7AM	0.25kwh	3.0kwh
PHEV	9.9kwh	8PM-7AM	0.3kwh	2.0kwh

Table 2 24 Hours Optimal Prices Offered by the Retailer.

Time	8AM	9AM	10AM	11AM	12PM	1PM	2PM	3PM
Price(cents)	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
Time	4PM	5PM	6PM	7PM	8PM	9PM	10PM	11PM
Price (cents)	12.00	12.00	12.00	14.00	14.00	14.00	14.00	14.00
Time	12AM	1AM	2AM	3AM	4AM	5AM	6AM	7AM
Price(cents)	14.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00

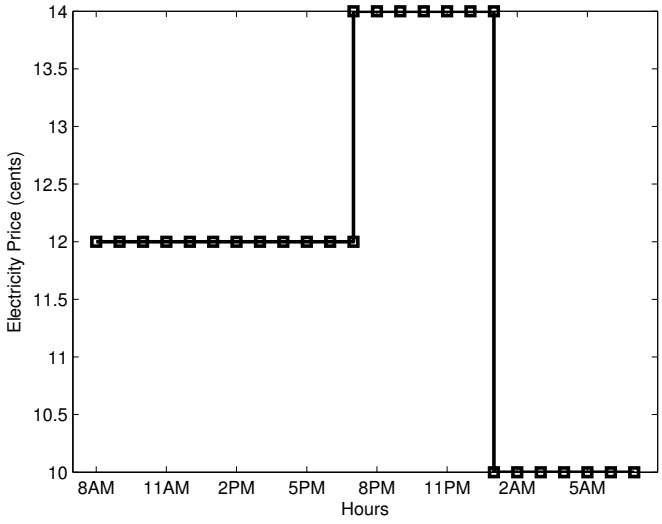


Figure 1 Optimal Real-time Prices Offered by the Retailer.

from 8 cents to 12 cents for mid-peak hours while the prices float between 6 cents to 10 cents for off-peak hours.

The aim of our proposed optimal RTP scheme is to find the optimal 24 hours prices by maximizing the retailer's profit (upper level problem). Besides this, with this identified price information, the customer can achieve his best benefit, i.e., minimize his payment bills (lower level problem).

Applying the open source solver YALMIP to our proposed bi-level programming problems, we can get the optimal 24 hours prices shown as Table 2 and Figure 1.

With the purpose to design a benchmark, we assume, without our proposed optimal appliances scheduling scheme, the appliances start working right at the beginning of the time interval H_s and at its typical power level. The energy consumption comparison of appliances with and without scheduling can be seen from Figure 2. We can easily find from Figure 2 that the customer shifts the energy use from high price periods to low price period. As a result, with our proposed scheduling scheme, the electricity bill of the customer for one day is reduced from 2.35 $ to 1.94 $.

Based on the above analysis, we can see that with this bi-level programming model not only the electricity retailer can maximize his benefits, but the customer can also benefit from the reduced electricity bills.

Last but not least, the proposed branch and bound algorithm is more efficient compared

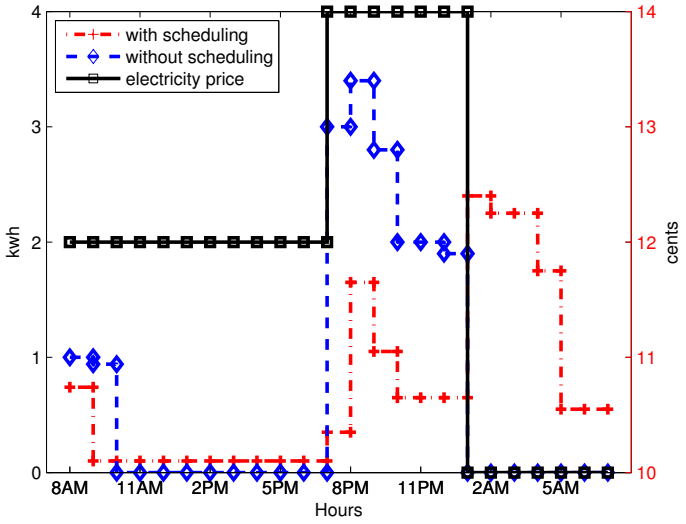


Figure 2 Energy Consumption Comparison with Scheduling and without Scheduling under Obtained Optimal Real-time Pricing.

with genetic algorithms used in [11]. Ten separate experiments for each approach have been done for the computation time comparison. The average time cost of the genetic algorithm approach in obtaining the optimal solution to our proposed bi-level programming model is around 120 seconds while the branch and bound algorithm takes only around 8 seconds.

5 Conclusion and Future Work

We propose a bi-level programming approach to model the interactions between the retailer and its customer. First, a electricity bill minimization model (lower level model) has been proposed for the customer to incentive him to change his electricity use pattern. Second, a profit maximization model (upper level model) for the retailer has been modelled. A branch and bound algorithm is chosen to solve this propose bi-level programming problem. As the simulation results show that both the retailer and the customer can benefit from the proposed framework, it has great potential to improve the implementation of current energy pricing programs, help customers to reduce the increasing energy bills, and change their energy usage patterns.

This work can be extended in several directions. First, we will enrich the lower level problem by considering the trade-off between minimizing bills and satisfying customer's comfort. Second, we will extend the current one-leader one follower case to one-leader multiple-followers bi-level programming model in our future work.

References

1 Jonathan F Bard and James T Moore. A branch and bound algorithm for the bilevel programming problem. *SIAM Journal on Scientific and Statistical Computing*, 11(2):281–292, 1990.

2 S. Bu, F. Richard Yu, and Peter X. Liu. A game-theoretical decision-making scheme for electricity retailers in the smart grid with demand-side management. In *2011 IEEE*

International Conference on Smart Grid Communications (SmartGridComm), pages 387–391, 2011.

3 C. Chen, S. Kishore, and L.V. Snyder. An innovative rtp-based residential power scheduling scheme for smart grids. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5956–5959. IEEE, 2011.

4 Jiang Chen, Bo Yang, and Xinping Guan. Optimal demand response scheduling with stackelberg game approach under load uncertainty for smart grid. In *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, pages 546–551. IEEE, 2012.

5 Benoît Colson, Patrice Marcotte, and Gilles Savard. Bilevel programming: A survey. *4OR*, 3(2):87–107, 2005.

6 Seppo Kärkkäinen Corentin Evens. Pricing models and mechanisms for the promotion of demand side integration. Technical Report VTT-R-06388-09, VTT Technical Research Centre of Finland, 2009.

7 Martine Labbé and Alessia Violin. Bilevel programming and price setting problems. *4OR*, pages 1–30, 2013.

8 Na Li, Lijun Chen, and Steven H. Low. Optimal demand response based on utility maximization in power networks. In *2011 IEEE Power and Energy Society General Meeting*, pages 1–8, 2011.

9 Johan Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

10 Huina Mao, Xiao-Jun Zeng, Gang Leng, Yong-Jie Zhai, and John A Keane. Short-term and midterm load forecasting using a bilevel optimization model. *Power Systems, IEEE Transactions on*, 24(2):1080–1090, 2009.

11 Fan-Lin Meng and Xiao-Jun Zeng. A stackelberg game approach to maximise electricity retailer's profit and minimse customers' bills for future smart grid. In *Computational Intelligence (UKCI), 2012 12th UK Workshop on*, pages 1–7. IEEE, 2012.

12 A Mohsenian-Rad and VWS Wong. Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. *Smart Grid, IEEE*, 1(3):320–331, 2010.

13 Amir-Hamed Mohsenian-Rad and Alberto Leon-garcia. Optimal Residential Load Control With Price Prediction in Real-Time Electricity Pricing Environments. *Smart Grid, IEEE*, 1(2):120–133, 2010.

14 Erling Pettersen, Andrew B Philpott, and Stein W Wallace. An electricity market game between consumers, retailers and network operators. *Decision support systems*, 40(3):427–438, 2005.

15 Vesna Radonjić and Vladanka Aćimović-Raspopović. Responsive pricing modeled with stackelberg game for next-generation networks. *annals of telecommunications-annales des télécommunications*, 65(7-8):461–476, 2010.

16 Pedram Samadi, Amir-Hamed Mohsenian-Rad, Robert Schober, Vincent W. S. Wong, and Juri Jatskevich. Optimal Real-Time Pricing Algorithm Based on Utility Maximization for Smart Grid. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 415–420, 2010.

17 Huijun Sun, Ziyou Gao, and Jianjun Wu. A bi-level programming model and solution algorithm for the location of logistics distribution centers. *Applied Mathematical Modelling*, 32(4):610–616, 2008.

18 Peng Yang, Gongguo Tang, and Arye Nehorai. A game-theoretic approach for optimal time-of-use electricity pricing. *Power Systems, IEEE Transactions on*, 28(2):884–892, 2013.

Dreaming Machines: On multimodal fusion and information retrieval using neural-symbolic cognitive agents

Leo de Penning[1], Artur d'Avila Garcez[2], and John-Jules C. Meyer[3]

1 **TNO Behaviour and Societal Sciences**
Soesterberg, The Netherlands
leo.depenning@tno.nl

2 **Department of Computing, City University**
London, UK
aag@soi.city.ac.uk

3 **Department of Information and Computing Sciences, Utrecht University**
Utrecht, The Netherlands
jj@cs.uu.nl

Abstract

Deep Boltzmann Machines (DBM) have been used as a computational cognitive model in various AI-related research and applications, notably in computational vision and multimodal fusion. Being regarded as a biological plausible model of the human brain, the DBM is also becoming a popular instrument to investigate various cortical processes in neuroscience. In this paper, we describe how a multimodal DBM is implemented as part of a Neural-Symbolic Cognitive Agent (NSCA) for real-time multimodal fusion and inference of streaming audio and video data. We describe how this agent can be used to simulate certain neurological mechanisms related to hallucinations and dreaming and how these mechanisms are beneficial to the integrity of the DBM. Finally, we will explain how the NSCA is used to extract multimodal information from the DBM and provide a compact and practical iconographic temporal logic formula for complex relations between visual and auditory patterns.

1998 ACM Subject Classification I.2.0 Cognitive simulation

Keywords and phrases Multimodal fusion, Deep Boltzmann Machine, Neural-Symbolic Cognitive Agent, Dreaming, Hallucinations

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.89

1 Introduction

The human brain has always inspired many of us to investigate and try to understand its complex processes. Ranging from neuroscientists that try to model the brain in terms of neurons, synapses and pathologies, to psychologists and cognitive scientists that try to model it in terms of human and social behaviour, to computer scientists that try to model it in terms of computational models that can perform intelligent tasks. A common tool in all these sciences is the use of abstract models of the human brain that help us to simulate, analyse and understand how it works. From a computer science perspective, computational models of the human brain are often based on models from neuroscience (e.g. neural networks) or models from cognitive and social sciences (e.g. cognitive models). These models have enabled computer scientists to build very complex systems that are able to perform tasks of human intelligence (e.g. visual recognition, speech recognition and driving a car). On the

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 89–94

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

other hand, these computational models have also been used in neural, cognitive and social sciences to investigate the human brain itself. For example, computational models have been used to investigate biological pathways in the visual cortex [5], neurological pathologies that cause hallucinations [10], the role of long-term memory in perception [8], and social dynamics of cognitive and affective processes [14]. Even in the investigation of more elusive and abstract processes related to the brain, computational models have been used. For example, to illustrate the role of mind and brain in cognitive psychology [12] and to explain the function of dreaming [1].

In this paper we will describe the use of a Deep Boltzmann Machine (DBM) as computational model to simulate certain neurological processes related to hallucination and dreaming and describe how these processes can be applied to multiple modalities, specifically streaming audio and video. Furthermore we will describe and illustrate how a Neural-Symbolic Cognitive Agent (NSCA) can be used to retrieve information from this model in a temporal logic formula that incorporates iconographic representations of the visual and auditory patterns.

2 Multimodal Deep Learning

Similar to the approach described in [9] we apply a Deep Boltzmann Machine (DBM) for multimodal fusion of visual and auditory information. A DBM can learn hierarchical representations of data, using several layers of Restricted Boltzmann Machines (RBMs) [11]. Each RBM represents a stochastic neural network with visible units $\mathbf{v}$, that represent input variables (or hidden-unit activations of lower-layer RBMs), and hidden units $\mathbf{h}$, that represent the likelihood of certain activation patterns in $\mathbf{v}$. There are symmetric weighted connections between the hidden and visible units with weights W, but no connections within the hidden units or visible units. The weights can be trained to model a joint probability distribution over $\mathbf{h}$ and $\mathbf{v}$ (Equation 1, where $\mathbf{b}$ and $\mathbf{c}$ denote the biases of the hidden and visible units and $\sigma(x)$ the logistic sigmoid function). This particular configuration makes it easy to compute the conditional probability distributions, when $\mathbf{v}$ or $\mathbf{h}$ is fixed (Equation 2), enabling the reconstruction of input data based on partial information in $\mathbf{v}$. This is done by sampling the conditional probability distribution in Equation 2, where $h'_j = 1$ with $p(h_j|\mathbf{v})$ (and $h'_j = 0$ otherwise), and calculating the reconstructed data $\mathbf{v'}$, where $v'_i = p(v_i|\mathbf{h'})$.

$$-\log P(\mathbf{v},\mathbf{h}) \propto E(\mathbf{v},\mathbf{h}) = -\mathbf{c}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{h}^T W\mathbf{v} \tag{1}$$

$$p(h_j|\mathbf{v}) = \sigma(b_j + w_j^T\mathbf{v}) \tag{2}$$

To train a DBM, each RBM layer is trained separately using Contrastive Divergence learning [5]. This learning algorithm tries to minimize the difference between $\mathbf{v}$ and $\mathbf{v'}$ by changing the weights using a Hebbian-like learning rule such that $\triangle W \cong \mathbf{v}\cdot\mathbf{h} - \mathbf{v'}\cdot\mathbf{h'}$, with the network in the long run learning to approximate the joint probability distribution $P(\mathbf{v},\mathbf{h})$.

Figure 1 depicts the RBMs used to model the auditory and visual patterns (1a, b) and two possible configurations for fusion of these patterns (1c, d). In this work we apply the same architecture as the bimodal DBN[1] (1d), to optimize the learning of relations across modalities (see [9]). We do not apply the deep autoencoders as proposed in [9] as we assume both modalities will be present during training and testing. Also, we will explain how

[1] Deep Boltzmann Machine are also referred to as Deep Belief Networks (DBN).

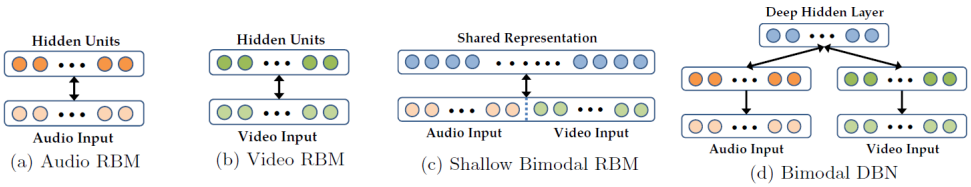


Figure 1 RBMs that model auditory (a) and visual patterns (b) and combine these patterns in higher-order multimodal representations (c, d).

certain neurological mechanisms can be used to overcome the multimodal inference problem addressed in [9]. To train the RBMs for audio (1a) and video (1b) we decode the audio stream as a spectrogram of 10 frames x 1024 frequencies using Discrete Cosine Transformation (DCT) on decoded audio samples, and the video stream as monochrome images that are reduced in scale, resulting in 160x120 pixels. Both transformations are fast and reversible allowing us to reconstruct video and audio from the DBM in real-time. As explained later, this approach will also enable us to do multimodal information retrieval and demonstrate the effect of neurological processes related to hallucinations and dreaming. As an extension to the DBM we also investigated the use of Recurrent Temporal RBMs in the top layer to model temporal sequences of audio and video patterns by taking into account the hidden unit activations in the previous time step [13].

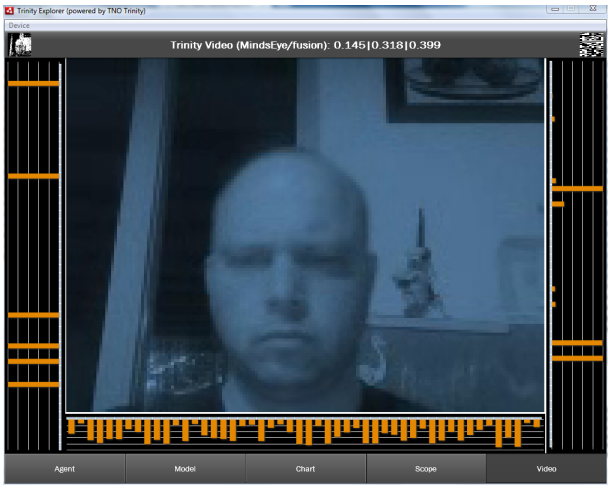


Figure 2 Adobe Flash based client that records from webcam and plays back reconstructed audio and video from a DBM. The real-time activations of all hidden units in the DBM is visualized as follows: on the left-side is depicted the hidden unit activations of the video, on the right-side the hidden unit activations of the audio, and on the bottom the hidden unit activations after multimodal fusion in the top layer of the DBM.

For demonstration purposes we implemented the DBM in a multi-agent platform, called Trinity[2], that supports real-time media streaming for Adobe Flash based clients that stream audio and video from a webcam. We implemented the DBM as part of a NSCA that enables the interpretation and reconstruction of audio and video information in a stream

[2] Trinity is a successor of the SimSCORM platform that has been developed for automated training and assessment [4].

and supports automated video indexing or assessment of observed human behaviour. As depicted in Figure 2, the client plays back the reconstructed audio and video and displays the real-time activation of the hidden units residing in each hidden layer. The use of a NSCA also allows to retrieve and investigate the contents of each visual, auditory and multimodal pattern that has learned by the DBM (see section 4). These patterns can be visualized in the tool by clicking on the bar of a related hidden unit in the activation graphs. We believe that this tool can help in future work on both multimodal fusion as well as the investigation of neurological processes.

3 Hallucinations and Dreaming

As described in [10], the DBM is a biologically plausible computational model for the investigation of neurological processes related to cortical learning, perception and diseases. It is able to simulate certain cortical processes that result in hallucinations due to loss of vision (i.e. Charles Bonnet Syndrome). Reichert describes how homoeostatic mechanisms in the cortex can stabilize neuronal activity to recover correct internal representations from degraded input. After some period this process can lead to complex vivid visual hallucinations. This mechanism can be implemented in the DBM as a regularization term for hidden unit biases (Equation 3) that is similar to mechanisms employed in other DBM-like models to enforce sparsity in the activations [9, 7].

$$\triangle b_i = \eta(p_i - a_i) \tag{3}$$

Using this model and regularization term, we have conducted several experiments that indeed demonstrate the forming of hallucinations when visual input is completely or partially blanked (mimicking loss of vision). These experiments also showed that when random noise is applied to the input, smaller overall bias shifts were necessary to restore original activity levels and produce hallucinations. This effect resembles another cognitive process, called reverse learning.

Reverse learning is a mechanism that is believed to be used in Rapid Eye Movement (REM) sleep to remove certain undesirable modes of interaction in networks of cells in the cerebral cortex. According to [1] the trace in the brain of the unconscious dream causes these modes to be weakened by applying random stimulation of the forebrain generated by the brain stem. This will tend to excite the inappropriate modes of brain activity, especially those which are too prone to be set off by random noise rather than by highly structured specific signals. Due to the random noise, overall neuron activity will drop, similar to the overall bias shift described before, automatically weakening the connections that encode these inappropriate modes.

Basically this means that reverse learning can also be regarded as a form of homoeostasis, which is beneficial to the integrity of the human brain and can be implemented in a DBM using the same stabilization mechanism as described before. We have implemented these mechanisms in all hidden layers of our DBM, resulting in a form of multimodal hallucination and dreaming. The effect of these mechanisms on the quality of the knowledge encoded in the DBM is still under investigation, but preliminary results have shown that the DBM indeed recovers from loss of audio or video input, producing hallucinations during stabilization of neuron activity, and that sparsity has improved the overall quality of the temporal relations encoded in the model.

4 Multimodal Information Retrieval

As described in [9], DBMs can produce good models for multimodal inference. For example, for the reconstruction of phonemes based on a visual representation of the mouth, and vice versa. But this approach will not explain the complex temporal relations encoded in a multimodal DBM. With a NSCA we are able to use an extraction mechanism that allows us to describe these multimodal relations, for example in terms of logic-based rules. As described in [3, 2], a NSCA uses the conditional probability distributions of a RBM to extract logic-based rules that describe the temporal relations between beliefs B encoded by the visible units and hypotheses H encoded by the hidden units. Typically, the temporal relations are represented by clauses of the form $H_1 \leftrightarrow B_1 \wedge B_3 \wedge \bullet H_1$ which denotes that hypothesis H_1 holds at time t if and only if beliefs B_1 and B_3 hold at time t and hypothesis H_1 holds at time $t-1$, where we use the previous time temporal logic operator $\bullet$ to denote $t-1$ [6]. If we extend this approach to a DBM we get clauses that describe hierarchical relations between hypotheses $H^{(l)}$ and lower-order hypotheses $H^{(l-1)}$. If we apply this notation to our multimodal DBM for audio and video we get clauses that describe higher-order temporal relations between auditory and visual patterns, such as $H_1^{fusion} \leftrightarrow H_1^{audio} \wedge H_4^{video} \wedge \bullet H_2^{fusion}$, and lower-order relations describing the most likely auditory and visual patterns in terms of pixels and frequencies, such as $H_4^{video} \leftrightarrow B_{10}^{video} \wedge B_{443}^{video} \wedge B_{753}^{video} \wedge ....$

Such textual descriptions would of course be very elaborate and impractical to understand at the level of individual pixels or frequencies. Therefore, we have implemented an iconographic representation for these visual and auditory patterns that enables us to present more compact and meaningful descriptions of H^{video} and H^{audio}. Similar to the approach suggested in [5], to investigate the weights of a RBM in terms of 2D images, we create icons from the pixel and frequency patterns that are extracted for each hypothesis H_j^{video} and H_k^{audio} and resample them in black and white to emphasize the most significant aspects of the patterns. An example, extracted during one of the experiments, of an iconographic temporal logic description of a multimodal relation is given in Equation 4. The first two icons show a person on the left side of the camera with a hand under his head. The other two icons visualize spectrograms of 10 frames x 1024 frequencies depicting the word "hel-lo" in phonemes.

$$H_{42}^{mind} \leftrightarrow \text{  } \wedge \text{  } \wedge \text{  } \wedge \text{  } \wedge \bullet H_7^{mind} \tag{4}$$

5 Conclusions and Future Work

Computational models used in AI research, such as the RBM and DBM, are becoming popular instruments in neural, cognitive and social sciences for the investigation of the human brain. In this paper, we discussed how these instruments can be used to model and simulate certain neurological processes, related to hallucinations and dreaming, such as homoeostasis and reverse learning. We have explained how such processes are beneficial to the recovery of appropriate and the reduction of inappropriate traces of the brain and implemented these mechanisms in a multimodal DBM for streaming audio and video. Early experiments with the DBM have shown similar effects as in homoeostasis and reverse learning (i.e. multimodal hallucinations and dreaming) and we expect these mechanisms will improve the integrity of the model, by stimulating sparsity, recovery of missing input, and unlearning inappropriate relations.

As part of future work we will investigate the actual improvements to the overall quality of the model, using benchmarks for comparison with other models, but also using the knowledge extracted from our model for expert analysis. In preparation of this, we already implemented the multimodal DBM as part of a NSCA that is able to extract temporal relations between auditory and visual patterns in the form of a iconographic temporal logic formula. Such a representation makes it practical to describe the visual and auditory patterns in terms of images and spectrograms. This will help us to understand and investigate the complex temporal relations encoded in multimodal DBMs and explain why certain neurological phenomenon occur, either in the DBM as a computational model or in the human brain that it tries to simulate.

References

1 F Crick and G Mitchison. The function of dream sleep. *Nature*, 304(5922):111–114, 1983.

2 Leo de Penning, Artur S. d'Avila Garcez, Luís C. Lamb, and John-Jules C. Meyer. A Neural-Symbolic Cognitive Agent for Online Learning and Reasoning. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, Barcelona, Spain, 2011.

3 Leo de Penning, R.J.M. den Hollander, H. Bouma, G.J. Burghouts, and A.S d'Avila Garcez. A Neural-Symbolic Cognitive Agent with a Mind's Eye. In *Workshop on Neural-Symbolic Learning and Reasoning at AAAI*, 2012.

4 Leo de Penning, Bart Kappé, and Eddy Boot. Automated Performance Assessment and Adaptive Training for Training Simulators with SimSCORM. In *Proc. of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, pages 1–7, Orlando, USA, 2009.

5 Geoffrey E Hinton. Learning to represent visual input. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 365(1537):177–84, January 2010.

6 Luís C. Lamb, R.V. Borges, and Artur S. d'Avila Garcez. A connectionist cognitive model for temporal synchronisation and learning. In *Proc. of the AAAI Conference on Artificial Intelligence*, pages 827–832. AAAI Press, 2007.

7 Honglak Lee, C Ekanadham, and A Ng. Sparse deep belief net model for visual area V2. *Advances in Neural Information Processing Systems*, 20, 2008.

8 Martial Mermillod, Robert M. French, Paul C. Quinn, and Denis Mareschal. The importance of long-term memory in infant perceptual categorization. In *Proc. of the 25th Annual Conference of the Cognitive Science Society*, Boston, Massachusetts, 2003.

9 Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal Deep Learning. In *International Conference on Machine Learning (ICML)*, Bellevue, WA, USA, 2011.

10 D.P. Reichert, Peggy Series, and A.J. Storkey. Hallucinations in Charles Bonnet Syndrome Induced by Homeostasis: a Deep Boltzmann Machine Model. *Advances in Neural Information Processing Systems*, 23(23):2020–2028, 2010.

11 Ruslan Salakhutdinov and Geoffrey E. Hinton. Deep boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

12 E. Smith and S. Kosslyn. *Cognitive Psychology: Mind and Brain*. Prentice-Hall, 2006.

13 Ilya Sutskever. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

14 Chantal Natalie van der Wal. *Social Agents: Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes*. PhD thesis, Vrije Universiteit Amsterdam, 2012.

Self-composition by Symbolic Execution

Quoc-Sang Phan

Queen Mary University of London
qsp30@eecs.qmul.ac.uk

Abstract

Self-composition is a logical formulation of *non-interference*, a high-level security property that guarantees the absence of illicit information leakages through executing programs. In order to capture program executions, self-composition has been expressed in Hoare or modal logic, and has been proved (or refuted) by using theorem provers. These approaches require considerable user interaction, and verification expertise. This paper presents an automated technique to prove self-composition. We reformulate the idea of self-composition into comparing pairs of symbolic paths of the same program; the symbolic paths are given by Symbolic Execution. The result of our analysis is a logical formula expressing self-composition in first-order theories, which can be solved by off-the-shelf Satisfiability Modulo Theories solvers.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Information Flow, Symbolic Execution, Satisfiability Modulo Theories

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.95

1 Secure information flow: from *non-interference* to *self-composition*

Information flow in an information theoretical context is the transfer of information from a variable H to a variable O in a given process. The simplest case of information flow is *explicit flow* (or direct flow) where the whole or partial value of H is copied directly to O, for example:

```
O = H + 3;
```

There are more subtle cases, which are categorized as *implicit flow* (or indirect flow). Consider, for example, the program below, which simulates a common password checking procedure:

Listing 1 a password checking program

```
if (H == L)
    O = true;
else
    O = false;
```

H is the password, i.e. the confidential data; L is the public input provided by the user; O is the observable output, $O = true$ means the password is accepted. Although H is not directly copied to O, there is still information flow leaked $H \rightarrow O$. This information is "small", but one can reveal all information about H if he is allowed to make enough attempts.

Obviously, information flow from confidential data to observable output is not desirable, which is the motivation of research in *secure information flow*. Dating back to the pioneering work of the Dennings in the 1970s [4], secure information flow analysis has been an active research topic for the last four decades.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 95–102
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
ICCSW

Non-interference. A popular security policy that guarantees the absence of information flow leaks is non-interference [2, 5]. It is stated as follows: suppose a program P takes secret input H, public input L and produces public output O. Then P satisfies the non-interference property iff the value of O does not depend on H.

There has been a large body of work that has used type systems for validating non-interference, following the idea of Volpano et al. [10]. Type systems are fast and the analysis is *safe*, which means a program is classified as "*secure*", then it is actually secure, there are no false negatives. However, they also return too many false positives, which means secure programs can be classified as "*insecure*". For example, consider again the two examples with a small modification to make them satisfy non-interference:

Listing 2 A trivial secure program

```
O = H - H + 3;
```

Listing 3 An "always-reject" password checking program

```
if (H == L)
    O = false;
else
    O = false;
```

Typing rules would always classify programs like the above as insecure. Another tricky case is of programs that leak information in the intermediate states, but sanitize information at the end, for example:

```
O = H + 3;
O = 3;
```

Given that the attacker can only observe the final value of the output O, the program is secure. However, it would be classified as insecure by type systems.

Self-composition. Another prominent approach for secure information flow is to use theorem proving, in which non-interference is logically formulated as *self-composition* [3, 1], as non-interference itself is not a logical property.

We assume a similar setting as in the case of non-interference: given a program P that takes secret input H, public input L and producing public output O, we denote by P_1 the same program as P, with all variables renamed: H as H_1, L as L_1 and O as O_1. For example, consider again the password checking program P in Listing 1, the composition of P and its copy P_1 is as follows:

```
if (H == L)
    O = true;
else
    O = false;
/* copy of the same program with all variables renamed */
if (H1 == L1)
    O1 = true;
else
    O1 = false;
```

Self-composition is expressed in Hoare-style framework as [1]:

$$\{L = L_1\}P; P_1\{O = O_1\} \tag{1}$$

The Hoare triple states that if the precondition $L = L_1$ holds, then after the execution of $P; P_1$, the postcondition $O = O_1$ also holds. Recall that non-interference requires the output O not to depend on the secret input H, which means that for any pair of possible executions of P that only differ in H, they have to agree on the public output O. In self-composition, the purpose of having the copy P_1 with all variables renamed is to have *another* P to compare with P, so self-composition is logical formulation of non-interference.

For the example above, by choosing $H = L \wedge H_1 \neq L_1$, it is easy to find a counterexample for the Hoare triple in (1), such that $L = L_1$ holds and $O = O_1$ does not hold. Therefore, the password checking program violates self-composition, and hence there is information leaked $H \rightarrow O$.

Compared to type system approach, the theorem proving approach is much more precise, returning no false positives. However, it is impractical in reality, as elegantly put in [9] by Terauchi and Aiken:

"When we actually applied the self-composition approach, we found that not only are the existing automatic safety analysis tools not powerful enough to verify many realistic problem instances efficiently (or at all), but also that there are strong reasons to believe that it is unlikely to expect any future advance."

Terauchi and Aiken also pointed out that the limitations of self-composition come from the symmetry and redundancy of the self-composed program, which lead to some partial-correctness conditions that hold between P and P_1. To find these conditions is crucial for the effectiveness of the analysis, however, finding them is in general impractical.

Moreover, to prove (or refute) self-composition with theorem provers requires considerable user interaction and verification expertise [3].

Contribution. This paper presents an automated technique for non-interference based on self-composition. The self-composition approach can be divided into two steps: first, to compose the program with a copy of itself; second, to perform analysis on the self-composed program. Our approach is to delay self-composing to the second step: first, we perform analysis on the original program with Symbolic Execution; second, we self-compose the result of the analysis to get the formula of self-composition. The idea of self-composition is to have a copy P_1 of P to compare with itself. We expand this idea into comparing all pairs of executions ρ of P and ρ_1 of P_1. Since it is impossible to enumerate all possible executions, we use Symbolic Execution to synthesize the symbolic paths that represents a set of concrete executions, and perform comparison on these symbolic paths, which we formulate as *path-equivalence*.

The delay of self-composing after performing the analysis is the main novelty of our approach. In this way, we could avoid the symmetry and redundancy of the self-composed program. Moreover, the symbolic paths synthesized by Symbolic Execution are presented by first-order theories, just as the generated formula of self-composition. The validity of this formula can be automatically and efficiently checked by powerful Satisfiability Modulo Theories (SMT) solvers.

2 Preliminaries

A deterministic program is modelled as a transition system:

$$P = (\Sigma, I, F, T)$$

where Σ is the set of program states; $I \subseteq \Sigma$ is the set of initial states; $F \subseteq \Sigma$ is the set of final states; and $T \subseteq \Sigma \times \Sigma$ is the transition function. Under this setting, a trace of (concrete)

execution of program P is represented by a sequence of states:

$$\rho = \sigma_0\sigma_1..\sigma_n$$

such that $\sigma_0 \in I, \sigma_n \in F$ and $\langle \sigma_i, \sigma_{i+1} \rangle \in T$ for all $i \in \{0, .., n-1\}$. We define two functions *init* and *fin* to get the initial state and final state of ρ:

$$\mathit{init}(\rho) = \sigma_0 \text{ and } \mathit{fin}(\rho) = \sigma_n$$

The semantics of P is then defined as the set $\mathcal{R}$ of all possible traces.

We assume that each initial state $\sigma \in I$ is a pair $\langle H, L \rangle$, i.e. $I = I_H \times I_L$, in which H is the confidential component to be protected and L is the public component that may be controlled by an attacker.

Symbolic Execution. Symbolic Execution (SE), first introduced by King in the 1970s [6], is a technique widely used in verification and testing. The key idea is the following. Instead of taking inputs to be concrete values, SE takes inputs to be symbols, e.g. α, β, which represent sets of concrete input values; the program is then executed just like in normal execution. In the setting of SE, the program P is modelled as a transition system:

$$P = (\Sigma^s, I^s, F^s, T^s)$$

where Σ^s is the set of symbolic states; each $\sigma^s \in \Sigma^s$ represents a set of concrete states $\sigma \in \Sigma$. $I^s \subseteq \Sigma^s$ is the set of initial symbolic states; $F^s \subseteq \Sigma^s$ is the set of final symbolic states; and $T^s \subseteq \Sigma^s \times \Sigma^s$ is the transition function. A symbolic path (symbolic trace) of the program P is represented by a sequence of symbolic states:

$$\rho^s = \sigma_0^s\sigma_1^s..\sigma_n^s$$

such that $\sigma_0^s \in I^s, \sigma_n^s \in F^s$ and $\langle \sigma_i^s, \sigma_{i+1}^s \rangle \in T^s$ for all $i \in \{0, \ldots, n-1\}$. The symbolic semantics of P is then defined as the set of all symbolic paths $\mathcal{R}^s$, which is also called as the *symbolic execution tree*. Likewise, each $\rho^s \in \mathcal{R}^s$ represents a set of traces $\rho \in \mathcal{R}$.

We denote by $X|_y$ the value of the variable X at the state y. After symbolically executing the program P with initial input symbols $H = \alpha, L = \beta$, for each $\sigma_i^s \in F^s$, i.e. each leaf of the symbolic execution tree, we have a symbolic formula for the value of the output O in the symbolic environment:

$$O|_{\sigma_i^s} = f_i(\alpha, \beta)$$

Another product of SE is the path condition $pc_i \equiv c_i(\alpha, \beta)$ for σ_i^s to be reachable. Each pc_i corresponds to a symbolic path ρ_i^s. The following theorem was also proved by King [6]:

▶ **Theorem 1.**

$$\forall i, j \in [1, n] \wedge i \neq j . pc_i \wedge pc_j = \bot$$

We define the function *path* such that:

$$\mathit{path}(\rho_i^s) = pc_i$$

The output O can be considered as a result of the following function:

$$O = \left\{ \begin{array}{ll} f_1(\alpha, \beta) & \text{if } c_1(\alpha, \beta) \\ f_2(\alpha, \beta) & \text{if } c_2(\alpha, \beta) \\ \ldots & \ldots \\ f_n(\alpha, \beta) & \text{if } c_n(\alpha, \beta) \end{array} \right\} \tag{2}$$

Or the following always holds:

▶ **Corollary 2.**

$$\forall i \in [1,n].c_i(\alpha,\beta) \rightarrow O = f_i(\alpha,\beta)$$

f_i and c_i are in general combination of first-order theories, e.g. *linear arithmetic*, *bit vector* and so on. SE tools make use of off-the-shelf SMT solvers to check the satisfiability of c_i, and eliminate unreachable paths (which may appear in the control flow graph).

3 Self-composition by Symbolic Execution

To avoid the limitation of the theorem proving approach, we need to reformulate the self-composition formula into a simpler logic which does not contain the program P. This is made possible by using the trace semantics of programs.

3.1 Self-composition as path-equivalence

Given a program P that takes secret input H, public input L and producing public output O; P_1 is the same program as P, with all variables renamed: H as H_1, L as L_1 and O as O_1. The trace semantics of P and P_1 are $\mathcal{R}$ and $\mathcal{R}_1$ respectively.

▶ **Definition 3** (trace-equivalence)**.** The program P satisfies non-interference if:

$$\forall \rho \in \mathcal{R}, \rho_1 \in \mathcal{R}_1.L|_{init(\rho)} = L_1|_{init(\rho_1)} \rightarrow O|_{fin(\rho)} = O_1|_{fin(\rho_1)} \tag{3}$$

It is stated similarly to the Hoare triple in (1): for all possible pairs of traces ρ of P, and ρ_1 of P_1: if $L = L_1$ at the initial states, then $O = O_1$ at the final states. At this point, we have a formulation of self-composition that does not involve the programs P and P_1.

However, even with simple programs, it is impossible to compute all the traces. Our solution is to use trace-equivalence with SE. Recall that each symbolic path represents a set of traces, and it is possible to build a complete symbolic execution tree (here we only consider bounded programs). Following Corollary 2, trace-equivalence in the context of SE is redefined as follows:

▶ **Definition 4** (path-equivalence)**.** The program P satisfies non-interference if:

$$\forall \rho^s \in \mathcal{R}^s, \rho_1^s \in \mathcal{R}_1^s.(L|_{init(\rho^s)} = L_1|_{init(\rho_1^s)}) \wedge path(\rho^s) \wedge path(\rho_1^s) \rightarrow (O|_{fin(\rho^s)} = O_1|_{fin(\rho_1^s)}) \tag{4}$$

In this way, we have an SMT formula, i.e. a combination of first-order theories. This is the key novelty of our approach, since the formulation of self-composition in first-order theories enables us to solve it efficiently using off-the-shelf SMT solvers.

3.2 Path-equivalence generation

Suppose P is symbolically executed with $H = \alpha, L = \beta$. To simplify the formula, we choose the input symbols for P_1 as $H_1 = \alpha_1, L_1 = \beta$ so that $L|_{init(\rho^s)} = L_1|_{init(\rho_1^s)}$ is automatically satisfied. That means:

$$(H|_{init(\rho^s)} = \alpha) \wedge (L|_{init(\rho^s)} = \beta) \wedge (H_1|_{init(\rho_1^s)} = \alpha_1) \wedge (L_1|_{init(\rho_1^s)} = \beta)$$

Given the result of SE is a function of the output O as in (2), the path-equivalence in (4) can be rewritten as:

$$PE \equiv DF \wedge IF$$

where:

$$DF \equiv \bigwedge_{i=1}^{n} c_i(\alpha,\beta) \wedge c_i(\alpha_1,\beta) \rightarrow (f_i(\alpha,\beta) = f_i(\alpha_1,\beta)) \tag{5}$$

$$IF \equiv \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} c_i(\alpha,\beta) \wedge c_j(\alpha_1,\beta) \rightarrow (f_i(\alpha,\beta) = f_j(\alpha_1,\beta)) \tag{6}$$

DF checks the path-equivalence when both P and P_1 follow the same symbolic path, and thus it guarantees the absence of direct flows. On the other hand, IF checks the path-equivalence when P and P_1 follow different symbolic paths, and it guarantees the absence of implicit flows.

4 Case Studies

We illustrate the approach with some toy examples. Here we assume the same setting as above: a program P with confidential input H, public input L, and output O. SE executes P with input symbols $H = \alpha$ and $L = \beta$.

4.1 Implicit flow

Consider the password checking program in Listing 1. By SE, we have:

$$O = \left\{ \begin{array}{ll} true & \text{if } \alpha = \beta \\ false & \text{if } \alpha \neq \beta \end{array} \right\}$$

DF and DF are generated as follows:

$$DF \equiv (\alpha = \beta \wedge \alpha_1 = \beta \rightarrow true = true) \wedge (\alpha \neq \beta \wedge \alpha_1 \neq \beta \rightarrow false = false)$$
$$IF \equiv \alpha = \beta \wedge \alpha_1 \neq \beta \rightarrow true = false$$

It is trivial to prove that DF is valid and IF is invalid, and thus the program violates non-interference and leaks information via implicit flows.

4.2 No flow

Consider the modified version of the password checking procedure in Listing 3. By SE, we have:

$$O = \left\{ \begin{array}{ll} false & \text{if } \alpha = \beta \\ false & \text{if } \alpha \neq \beta \end{array} \right\}$$

DF and IF are generated as follows:

$$DF \equiv (\alpha = \beta \wedge \alpha_1 = \beta \rightarrow false = false) \wedge (\alpha \neq \beta \wedge \alpha_1 \neq \beta \rightarrow false = false)$$
$$IF \equiv \alpha = \beta \wedge \alpha_1 \neq \beta \rightarrow false = false$$

It is trivial to prove that both DF and IF are valid, and thus the program satisfies non-interference. Note that this is the case that type systems, taint analysis would decide as violating non-interference.

No confidential data involved. Consider again the password checking program, with a small modification to exclude the confidential data in its computation, i.e. to make it secure.

Listing 4 A program without confidential data

```
if (L == 3)
    O = true;
else
    O = false;
```

Similarly we have:

$$O = \left\{ \begin{array}{ll} true & \text{if } \beta = 3 \\ false & \text{if } \neg(\beta = 3) \end{array} \right\}$$

DF and IF are derived as:

$$DF \equiv (\beta = 3 \wedge \beta = 3 \rightarrow true = true) \wedge (\neg(\beta = 3) \wedge \neg(\beta = 3) \rightarrow false = false)$$
$$IF \equiv \beta = 3 \wedge \neg(\beta = 3) \rightarrow true = false$$

Both DF and IF are valid, which confirms the intuition that the program is secure.

4.3 Both implicit and explicit flows

Consider the following data sanitization program:

```
if (H < 16)
    O = H + L;
else
    O = L;
```

The summaries and path conditions returned by SE are as follows:

$$O = \left\{ \begin{array}{ll} \alpha + \beta & \text{if } \alpha < 16 \\ \beta & \text{if } \neg(\alpha < 16) \end{array} \right\}$$

DF and DF are generated similarly:

$$DF \equiv (\alpha < 16 \wedge \alpha_1 < 16 \rightarrow \alpha + \beta = \alpha_1 + \beta) \wedge (\neg(\alpha < 16) \wedge \neg(\alpha_1 < 16) \rightarrow \beta = \beta)$$
$$IF \equiv \alpha < 16 \wedge \neg(\alpha_1 < 16) \rightarrow \alpha + \beta = \beta$$

It is easy to find counterexamples to make DF and IF invalid, for example: $(\alpha = 1; \alpha_1 = 2)$ for DF and $(\alpha = 1; \alpha_1 = 17)$ for IF. So the program leaks via both implicit and explicit flows.

5 Related Work

Self-composition was first introduced by Darvas et al. [3] who expressed it in dynamic logic and proved information flow properties for Java CARD programs. Their approach is not automated, requiring users to provide loop invariants, induction hypotheses and so on. Barthe et al. [1] then coined the term "self-composition" and investigated its theoretical aspects, extending the problem to non-deterministic and termination-sensitive cases.

Terauchi and Aiken [9] found that self-composition was problematic, since the self-composed programs contains symmetry and redundancy. They proposed a type-directed transformation for a simple imperative language to deal with the problem. Milushev et al.

[7] implemented this type-directed transformation and used *Dynamic Symbolic Execution* (also known as *concolic testing*) as a program analysis tool for non-interference.

To our knowledge, our technique is unique in that it only performs analysis on the original program, rather than the self-composed program, the idea of self-composition is shown in the way we rename the symbolic formula, not in the analysis stage.

In our previous work [8], we proposed Symbolic Quantitative Information Flow (SQIF), an approach that uses Symbolic Execution to "measure" information flow leaks, i.e. quantitative information flow.

6 Conclusion

We have presented an automated method for secure information flow analysis. We build our work on the classical self-composition approach. However, instead of performing the analysis on the self-composed program, we use SE on the original program. This is enabled by reformulating self-composition into path-equivalence, a property for symbolic paths returned by SE.

Acknowledgements. We thank Nikos Tzevelekos and the anonymous reviewers for constructive comments.

References

1 Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, CSFW '04, pages 100–, Washington, DC, USA, 2004. IEEE Computer Society.

2 E. S. Cohen. Information transmission in sequential programs. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 297–335. Academic Press, 1978.

3 Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In *Proceedings of the Second international conference on Security in Pervasive Computing*, SPC'05, pages 193–209, Berlin, Heidelberg, 2005. Springer-Verlag.

4 Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, July 1977.

5 Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

6 James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, July 1976.

7 Dimiter Milushev, Wim Beck, and Dave Clarke. Noninterference via symbolic execution. In *Proceedings of the 14th joint IFIP WG 6.1 international conference and Proceedings of the 32nd IFIP WG 6.1 international conference on Formal Techniques for Distributed Systems*, FMOODS'12/FORTE'12, pages 152–168, Berlin, Heidelberg, 2012. Springer-Verlag.

8 Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, and Corina S. Păsăreanu. Symbolic quantitative information flow. *SIGSOFT Softw. Eng. Notes*, 37(6):1–5, November 2012.

9 Tachio Terauchi and Alex Aiken. Secure information flow as a safety problem. In *Proceedings of the 12th international conference on Static Analysis*, SAS'05, pages 352–367, Berlin, Heidelberg, 2005. Springer-Verlag.

10 Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4(2-3):167–187, January 1996.

Evaluation of Social Personalized Adaptive E-Learning Environments: End-User Point of View

Lei Shi, Malik Shahzad Awan, and Alexandra I. Cristea

University of Warwick
Department of Computer Science, University of Warwick, Coventry, CV4 7AL. UK
{lei.shi, malik, acristea}@dcs.warwick.ac.uk

Abstract

The use of adaptations, along with the social affordances of collaboration and networking, carries a great potential for improving e-learning experiences. However, the review of the previous work indicates current e-learning systems have only marginally explored the integration of social features and adaptation techniques. The overall aim of this research, therefore, is to address this gap by evaluating a system developed to foster social personalized adaptive e-learning experiences. We have developed our first prototype system, Topolor, based on the concepts of Adaptive Educational Hypermedia and Social E-Learning. We have also conducted an experimental case study for the evaluation of the prototype system from different perspectives. The results show a considerably high satisfaction of the end users. This paper reports the evaluation results from end user point of view, and generalizes our method to a component-based evaluation framework.

1998 ACM Subject Classification H.3.4 Systems and Software, H.5.1 Multimedia Information Systems, K.3.1 Computer Uses in Education

Keywords and phrases adaptive educational hypermedia, social e-learning, evaluation

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.103

1 Introduction

Adaptive Educational Hypermedia (AEH) [1] has been designed to offer a personalized learning process. It combines the ideas from Adaptive Hypermedia [2] and Intelligent Tutoring Systems [22], with the aim of producing applications wherein learning contents is adapted to personalized needs. The rapidly emerging social networking apps, offers an opportunity to improve the adaptive e-learning experience by introducing a social dimension. The study on introducing social dimension into adaptive e-learning has recently accumulated a considerable set of theories and techniques, which have been used for building a variety of e-learning systems with both adaptive and social features. As these theories and techniques promote such systems from research labs to real usage, the evaluation becomes more crucial and even more important than proposing new but questionable theories and techniques [5].

This paper aims to 1) present the evaluation of Topolor [18] that was designed based on the concepts of AEH [1] and Social E-Learning [9], and 2) propose a novel component-based evaluation framework that was generalized from our evaluation method. The evaluation was conducted from end user point of view, which reflect critical feedbacks during the real usage. It also intends to minimize the level of biasedness arising from the answers of a user.

In the following, section 2 presents related works, including existing e-learning systems that support adaptation and social interaction, and the existing evaluation methods; section 3 introduces Topolor; section 4 describes the collected data and questionnaire, and reports the evaluation results; section 5 summarizes our evaluation methods to propose a generic evaluation framework; section 6 draws conclusions and outlines future research.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 103–110

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Related Work

An AEH system aims at producing educational applications in which learning contents is adapted to every learner's personalized needs, such as knowledge level, learning goals and learning styles [3]. To extend an AEH system with social networking services, the support for knowledge networking and community building should be added to the system [8].

Several e-learning systems and services with both adaptive and social features have been developed in the last decade. For instance, Progressor [12] is a visual interface for open student modelling [13]. It provides students with a holistic and easy-to-grasp view on their progress and allows relating it to the progress of other students. QuizGuide [6] is an adaptive e-learning system that helps students in selecting relevant self-assessment quizzes assigned to topics. It uses adaptive annotation to support adaptive navigation, in order to provide personalized guidance to show which topics are more important and which require further study. Knowledge Sea II [4] uses social navigation to help students navigate from lectures to relevant online tutorials in a map style social navigation based on traffic and annotation.

These systems have only focused on a few aspects of adaptive and social facilities in an e-learning environment, thus, requiring the development of more comprehensive systems that support the integration and mutual promotion of these facilities. Further, the flourishing development of social, personalized and adaptive e-learning systems requires more generic approaches to evaluate and compare different systems. Therefore, it is essential to develop new evaluation methods that provide wider coverage in an e-learning environment.

One important issue associated with the development of a social personalized adaptive e-learning system is selecting an effective evaluation method to capture its broader perspective. It's also important that the adopted evaluation method is appropriate and correct [10]. Several evaluation methods have been developed. For example, Chao [7] suggested 5 criteria to such as 1) e-learning material, 2) quality of web learning platform, 3) synchronous learning, 4) self-learning, and 5) learning record. Ozkan [14] proposed a 6-dimension framework 1) system quality, 2) service quality, 3) content quality, 4) learner perspective, 5) instructor attitudes, and 6) supportive issues. These evaluation methods are only able to cover limited perspectives, although they were trying to cover more. Besides, they have no ability to compare, classify or linguistically represent different evaluated e-learning systems.

3 Topolor

Topolor [20] is featured as a social personalized adaptive e-learning system, and has been used as an online learning environment at the University of Warwick. Topolor was developed based on a classical layered architecture (inspired by the Dexter model [11]), extended with a social flavour: a storage layer, a persistence infrastructure for physical entities; and a runtime layer, parsing adaptation strategies to present adaptive and/or adaptable learning materials, providing Web 2.0 tools for social interaction, and monitoring learners' behaviour [15].

Topolor has a Facebook-like appearance (Fig. 1a): the profile avatar and user information, the fixed-position top menu, the left navigation bar, and the information flow wall for social interaction. As shown in Fig. 1b, Topolor supports topic and learning path adaptation, which provides various levels of granularity of learning content adaptation, and personalized order of learning those recommended topics; it supports peer recommendation, which is based on each learner's learning history and previous performance, in order to provide opportunities for them to learning from each other; it also support social interaction, which provides Web 2.0 tools that they are familiar with, so that they can comment on, share a topic, a question, a note, etc. This is a much broader range of adaptation than in regular AEH systems.

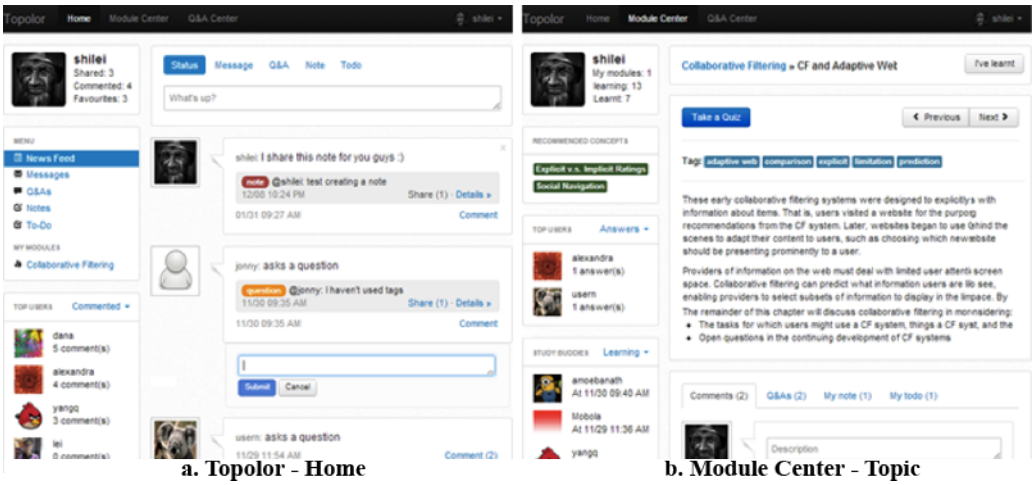


Figure 1 The screenshots of Topolor: a. Topolor – Home; b. Module Centre – Topic.

4 Evaluation

To evaluate Topolor, an experiment was conducted with the help of 21 students at the University of Warwick. They were learned an online course on "collaborative filtering" using Topolor in a 2-hour intensive learning session. Before the session, a 'to-do list' was handed out to make sure they have a reminder of all functionalities at their disposal. The order of using the functionalities, and if to repeat was up to them. During the session, a logging mechanism kept track of each of their actions. After the session, they were asked to fill in an optional and anonymous questionnaire. 10 out of 21 students returned the questionnaire.

Based on the collected data, Topolor has been evaluated from different perspectives [16][17][21]. We have also investigated learning behaviour patterns of users using machine learning, educational data mining and data visualization techniques [19]. This paper focuses on only the evaluation from the end user's point of view based on the questionnaire analysis.

This method consists of 3 perspectives: *functionality*, *learning perspective* and *system prospect*. The following sections present the analysis process and report evaluation results.

4.1 Functionality

10 functionalities were evaluated from two performance aspects: usefulness and ease of use as below. The score values of the two considered performance aspects of each functionality ranged between 0.85-1.40 and 0.95-1.44 for 'usefulness' and 'ease of use' respectively. Figure 2 presents the classification of them on a Likert Scale from -2 to 2.

Overall-Sub. This represents the overall view of each subsystem.

Status. The status (post) functionality supports learners to publish and share their learning status and comment on each other's status.

Messaging. It aims at helping in making the intra-system communication more efficient.

Q&A. The questioning and answering functionality helps learners learn and manage the queries related to learning topics.

Note. The note management functionality records learners' personal thought related to learning topics. Notes can also be shared to other learners.

To-do. The to-do management functionality aims at helping learners arrange their own learning plans and remind them to finish their tasks.

Module. The module management functionality includes activities such as arranging topics within a module, reviewing topics learnt, accessing to a recommended topic.

Topic. The represents the functionalities that supports topic-learning activities such as accessing the previous and next learning topic according to the recommended learning path, discussing with others who are learning the same topic, commenting on the topic.

Testing. This includes taking quizzes for a learning topic and taking tests for a module (a set of organized learning topics). It also assesses the process of reviewing quizzes/tests, and getting access to the learning topics related to the questions in a quiz/test.

Statistics. This represents the statistics about the numbers of, e.g., how many topics a learner has learnt, how many questions a learner has asked/answered, how many status (post) a learner has commented on and shared, and so on.

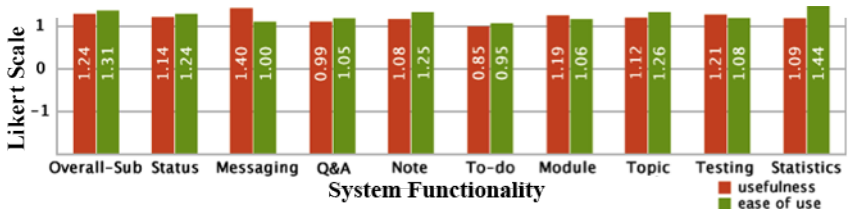


Figure 2 The result from the System Functionality Questionnaire.

4.2 Learning Perspective

The learning perspective was tested by the "Overall Topolor System – Five Scales for Believes" questionnaire, which include 7 questions shown as below. The individual score values for each of the questions are shown in Figure 3. **Q1.** I believe Topolor helps me learn more topics; **Q2.** I believe Topolor helps me learn more profoundly (deeply); **Q3.** I believe Topolor increases my learning outcome; **Q4.** I believe compare to other e-learning system, Topolor is easy to use; **Q5.** I believe compare to other e-learning system, Topolor is useful; **Q6.** I believe that the interaction with Topolor is easy to learn; **Q7.** I believe that the interaction with Topolor is easy to remember how to use.

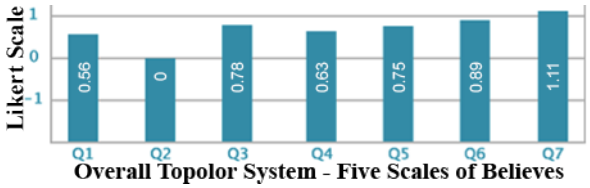


Figure 3 The result from the Overall Topolor System – Five Scales for Believes Questionnaire.

4.3 System Prospect

Two parameters were considered for evaluating the system prospects: 1) identification of the most useful features; and 2) improvements suggested by users. These two parameters were evaluated using a questionnaire with open-ended questions, in order to allow users to present a more practical and broader feedbacks on the features that were either helpful in learning or necessary to improve to be more useful features. From the questionnaire, four 'Best' features (Table 1) and four 'Need to Improve' features (Table 2) were identified respectively.

Table 1 The reported "Best Features".

Best Features	Frequency
Ask & answer questions	3
Filter learning content	3
Review quizzes	2
Discuss with others	2
Identified features: **4**	**10**

Table 2 Suggestions on improvements.

Improvements	Frequency
Share questions	4
To-do management	3
Interaction with contents	2
Searching tools	2
identified features: **4**	**11**

5 Evaluation Framework

We propose a component-based evaluation framework to generalize our methods that and overcome the shortcomings of existing methods. The proposed framework has 4 components. The first 3 evaluate an e-learning system from different perspectives. The 4th component combines the evaluation results of the first 3, summarizes and provides system classification and linguistic representation. The score values for the first 3 components are obtained from a questionnaire that system users fill in, after using the system for a given period of time.

5.1 Functionality

This component aims at using a Likert Scale to evaluate system functionalities from different performance aspects such as accessibility, effectiveness, operability, reliability, scalability and usability. We associate a weight, w with each considered performance aspect, which represents its significance. The score value of this component, C_{FUNC}, is calculated using Eq. 1a for representing the overall system value against the considered system functionalities. The score value is calculated by taking the weighted sum of the considered performance aspects, $SubSys_{(aspectID,subSysID)}$, and the associated weight, $w_{(aspectID,subSysID)}$, where *aspectID* represents a considered performance aspect; *subSysID* represents a considered sub-systems; m represents the number of the considered sub-systems. The generalized description of $SubSys_{(aspect,subSys)}$ for each considered system functionality represented as $F_{(aspectID,funcID,subSysID)}$, could be calculated using Eq. 1b, where *funcID* represents a considered functionality; $w_{(aspectID,funcID)}$ represents the corresponding associated weight; n represents the number of the considered system functionalities within the sub-system. $F_{(aspectID,funcID,subSysID)}$ could be calculated using Eq. 1c, where $q_{(i,j)}$ represents the j$^{\text{th}}$ question related to the considered system functionality, $F_{(aspectID,funcID,subSysID)}$, answered by the i$^{\text{th}}$ respondent; w_j represents the corresponding associated weight of this question; k represents the number of the questions related to the considered performance aspect of a considered system functionality; a represents the total number of respondents. The term *1/a* is used to minimize the level of biasedness arising from the answers of a respondent.

$$C_{FUNC} = \sum_{subSysID=1}^{m} SubSys_{(aspectID,subSysID)} \times w_{(aspectID,subSysID)} \tag{1a}$$

$$SubSys_{(aspectID,subSysID)} = \sum_{funcID=1}^{n} F_{(aspectID,funcID,subSysID)} \times w_{(aspectID,funcID)} \tag{1b}$$

$$F_{(aspectID,funcID,subSysID)} = \frac{1}{a} \times \sum_{i=1}^{a} \sum_{j=1}^{k} q_{(i,j)} \times w_j \tag{1c}$$

5.2 Learning Perspective

This component evaluates the impact that the learning system had on the overall learning experience as perceived by learners. It also evaluates the user-system interaction considered as helpful in learning as perceived by learners based on a Likert Scale against the defined criteria. The score value for this component, C_{LEARN} could be calculated using Eq. 2, where $q_{(j,j)}$ is the j$^{\text{th}}$ question answered by the i$^{\text{th}}$ respondent; w_j represents the corresponding associated weight of this question; k represents the number of the questions related to the considered performance aspect of a functionality; a represents the total number of respondents.

$$C_{LEARN} = \frac{1}{a} \times \sum_{i=1}^{a} \sum_{j=1}^{k} q_{(i,j)} \times w_j \tag{2}$$

5.3 System Prospect

This component identifies the learner characterization for the system features using simple questions grouped into different categories such as 1) identifying "the best feature(s)" of an e-learning system, 2) identifying the features that require further improvements. The score value for this component, 'C_{PROS}' could be calculated using Eq. 3a, where $Freq_j$ could either represent the number of features identified as "the best feature(s)" by a learner or those requiring improvements; w_j represents the associated significance of the feature category, i.e. the most useful and those requiring improvements; k represents the distinct feature categories in the evaluation. $Freq_j$ could be calculated by counting the frequency of an identified feature, $FeatureCount_i$, as grouped by the ith respondent, and a represents the total number of respondents, as shown in Eq. 3b. A value ranging from -2 to 2 represents the frequency of a reported feature. For 0 or 1 reported feature, $FeatureCount_i$ has an assigned value -2; Similarly, for 2, 3 and 4 reported features, the $FeatureCount_i$ has an assigned value of -1, 0 and 1 respectively. For 5 or more features, the assigned value is 2.

$$C_{PROS} = \frac{1}{k} \times \sum_{j=1}^{k} Freq_j \times w_j \tag{3a}$$

$$Freq_j = \frac{1}{a} \times \sum_{i=1}^{a} FeatureCount_i \tag{3b}$$

5.4 Overall System Classification and Linguistic Representation

This component provides an overall system classification and linguistic representation for the evaluated system. Its score value could be calculated using Eq. 4, where C_{FUNC}, C_{LEARN} and C_{PROS} have been calculated using Eq. 1a, Eq. 2 and Eq. 3a respectively, and then rounded off the mean score value to the nearest integer value on a Likert Scale from -2 to 2 (-2: *very bad*; -1: *bad*; 0: *medium*; 1: *good*; 2: *very good*).

$$System = \left\| \tfrac{1}{3} \times (C_{SYSFUNC} + C_{LEARN} + C_{PROS}) \right\| \tag{4}$$

The measurements in this component mainly involved counting the frequency of the system features reported by the learners against the two parameters considered in this case study. To maintain consistency in the value scale with the other two framework components,

namely, system functionality and learning perspective, we used the same scale, from -2 to 2, for assigning a score to each reported feature, and the score value is kept constant at 2 when the number of reported features exceeds 5. Likewise, for a non-reported feature or suggestion, a default score of -2 is added for maintaining consistency of results on the defined scale.

6 Conclusions and Future Work

In this paper, we have reported the evaluation of Topolor, a social personalized adaptive e-learning environment, and generalized the evaluation method to a component-based evaluation framework that provides broader and objective perspectives of a system evaluation report. The framework consists of 4 components, namely, *functionality*, *learning perspective*, *system prospect* and *system classification*. These components could be used for evaluating the effectiveness of a social personalized adaptive e-learning system from different perspectives with low level of biasedness. The evaluation involves the calculation of a score value using a component-specific mathematical model for each components. These score values are further used for calculating an overall rounded mean score value for the system. This rounded mean score value is then mapped to a Likert Scale for classifying the system into five (5) different categories, namely, 1) *very bad*, 2) *bad*, 3) *medium*, 4) *good* and 5) *very good*.

We further plan to broaden the data set for the evaluation of Topolor, so that, along with human experts, we could deploy advanced artificial intelligence techniques, e.g., artificial neural networks, case-based reasoning, machine learning, etc., for automatically learning the associated weights of framework components, system functionalities and question items during the system classification process. We further plan to extend the evaluation process by using fuzzy rule-based reasoning and consider the linguistic representation for classifying a social personalized adaptive e-learning system to present a more empirical view of system performance. Our future plans also include introducing mechanisms for estimating cost and effort associated with the measurement using our evaluation framework and perform a cost-and-benefit analysis for evaluating social personalized adaptive e-learning systems.

We claim that the proposed methodology is transferable to other social personalized adaptive e-learning systems and would enable researchers to: 1) gain a global view of assessments on an e-learning system; 2) evaluate each sub-system and functionality from different perspectives; 3) obtain learners perception of learning experience; 4) identify the best system features from learners' point of view and gain suggestions on the system improvement; 5) quantify the evaluation using component-specific mathematical models; and 6) classify the system on a Likert Scale for linguistic representation.

References

1 Peter Brusilovsky. Adaptive educational hypermedia. *International PEG Conference*, strony 8–12, 2001.

2 Peter Brusilovsky. Adaptive hypermedia. *User modeling and user-adapted interaction*, 11(1-2):87–110, 2001.

3 Peter Brusilovsky. Adaptive educational hypermedia: From generation to generation. *Proceedings of 4th Hellenic Conference on Information and Communication Technologies in Education, Athens, Greece*, strony 19–33, 2004.

4 Peter Brusilovsky, Girish Chavan, Rosta Farzan. Social adaptive navigation support for open corpus electronic textbooks. *Adaptive Hypermedia and Adaptive Web-Based Systems*, strony 24–33. Springer, 2004.

5 Peter Brusilovsky, Charalampos Karagiannidis, Demetrios Sampson. Layered evaluation of adaptive learning systems. *International Journal of Continuing Engineering Education and Life Long Learning*, 14(4):402–421, 2004.

6 Peter Brusilovsky, Sergey Sosnovsky, Olena Shcherbinina. Quizguide: Increasing the educational value of individualized self-assessment quizzes with adaptive navigation support. *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, wolumen 2004, strony 1806–1813, 2004.

7 Ru-Jen Chao, Yueh-Hsiang Chen. Evaluation of the criteria and effectiveness of distance e-learning with consistent fuzzy preference relations. *Expert Systems with Applications*, 36(7):10657–10662, 2009.

8 Mohamed Amine Chatti, Matthias Jarke, Dirk Frosch-Wilke. The future of e-learning: a shift to knowledge networking and social software. *International journal of knowledge and learning*, 3(4):404–420, 2007.

9 Stephen Downes. Feature: E-learning 2.0. *Elearn magazine*, 2005(10):1, 2005.

10 Cristina Gena, Stephan Weibelzahl. Usability engineering for the adaptive web. *The adaptive web*, strony 720–762. Springer-Verlag, 2007.

11 Frank Halasz, Mayer Schwartz, Kaj Grønbæk, Randall H Trigg. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.

12 I-Han Hsiao, Julio Guerra, Denis Parra, Fedor Bakalov, Birgitta König-Ries, Peter Brusilovsky. Comparative social visualization for personalized e-learning. *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 303–307. ACM, 2012.

13 Antonija Mitrovic, Brent Martin. Evaluating the effect of open student models on self-assessment. *Journal of Artificial Intelligence in Education*, 17(2):121–144, 2007.

14 Sevgi Ozkan, Refika Koseler. Multi-dimensional students' evaluation of e-learning systems in the higher education context: An empirical investigation. *Computers & Education*, 53(4):1285–1296, 2009.

15 Lei Shi, Dana Al Qudah, Alexandra I. Cristea. Designing social personalized adaptive e-learning. *The 18th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2013)*, 2013.

16 Lei Shi, Malik Shahzad K Awan, Alexandra I Cristea. Evaluating system functionality in social personalized adaptive e-learning systems. *Scaling up Learning for Sustained Impact*, strony 633–634. Springer, 2013.

17 Lei Shi, Dana Al Qudah, Alexandra I. Cristea. Social e-learning in topolor: a case study. *IADIS International Conference e-Learning*, 2013.

18 Lei Shi, Dana Al Qudah, Alaa Qaffas, Alexandra I Cristea. Topolor: a social personalized adaptive e-learning system. *User Modeling, Adaptation, and Personalization*, strony 338–340. Springer, 2013.

19 Lei Shi, Alexandra I. Cristea, Malik Shahzad Awan, Craig Stewart, Maurice Hendrix. Towards understanding learning behavior patterns in social adaptive personalized e-learning systems. *The 19th Americas Conference on Information Systems (AMCIS 2013)*, strony 708–711. Springer-Verlag, Berlin, Heidelber, 2013.

20 Lei Shi, George Gkotsis, Karen Stepanyan, Dana Al Qudah, Alexandra I. Cristea. Social personalized adaptive e-learning environment - topolor: Implementation and evaluation. *The 16th International Conference on Artificial Intelligence in Education (AIED 2013)*, strony 708–711. Springer-Verlag Berlin Heidelberg, 2013.

21 Lei Shi, Karen Stepanyan, Dana Al Qudah, Alexandra I. Cristea. Evaluation of social interaction features in topolor - a social personalized adaptive e-learning system. *The 13th IEEE International Conference on Advanced Learning Technologies (ICALT 2013)*, 2013.

22 Derek Sleeman, John Seely Brown. Intelligent tutoring systems. 1982.

Logical Foundations of Services

Ionuţ Ţuţu[1,2]

1 **Department of Computer Science, Royal Holloway University of London**
2 **Institute of Mathematics of the Romanian Academy, Research group of the project ID-3-0439**
ittutu@gmail.com

Abstract

In this paper we consider a logical system of networks of processes that interact in an asynchronous manner by exchanging messages through communication channels. This provides a foundational algebraic framework for service-oriented computing that constitutes a primary factor in defining logical specifications of services, the way models of these specifications capture service orchestrations, and how properties of interaction-points, i.e. points through which such networks connect to one another, can be expressed. We formalise the resulting logic as a parameterised institution, which promotes the development of both declarative and operational semantics of services in a heterogeneous setting by means of logic-programming concepts.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Formal methods, Service-oriented computing, Institution theory

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.111

1 Introduction

Service-oriented computing is a recent paradigm focusing on computation in data processing infrastructures that are globally available, and in which software applications can discover and bind dynamically to services offered by providers. The present paper builds on earlier theoretical work on algebraic structures that capture the way services are orchestrated [8, 6], and on the mechanisms that formalise the discovery of services that can be bound to a client application in terms of logical specifications of required/provided services [9, 7]. It explores one of the most technical aspects of a recent approach proposed in [3] that describes how aspects specific to the logic-programming paradigm can be used to capture the declarative and operational semantics of service-oriented computing.

To this purpose, we advance an integrated algebraic framework that constitutes the primary factor in defining logical specifications of services, as well as models of these specifications that correspond to orchestrations of components depending upon externally provided services. Our work upgrades the formalism considered in [3] by making a clear distinction between specifications of services and their orchestrations, which results in a framework that we consider to be more appropriate for addressing aspects such as heterogeneity.

Since the logic-programming semantics of services is technically based on an underlying logical system that is formalised as an institution [10], we will concentrate our efforts on proving that the proposed logic of networks of processes constitutes an institution. We recall that the theory of institutions is a categorical abstract model theory [4] that promotes a universal approach to the study of logics by abstracting the notion of truth, which is supposed to be invariant with respect to the change of notation. Formally, an *institution* is a quadruple $\mathcal{I} = \langle \mathbb{S}\mathrm{ig}^{\mathcal{I}}, \mathsf{Sen}^{\mathcal{I}}, \mathsf{Mod}^{\mathcal{I}}, \models^{\mathcal{I}} \rangle$ that consists of

– a category $\mathbb{S}\mathrm{ig}^{\mathcal{I}}$ of *signatures* and *signature morphisms*,

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 111–118

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- a *sentence functor* $\mathsf{Sen}^{\mathcal{I}}\colon \mathbb{S}\mathrm{ig}^{\mathcal{I}} \to \mathbb{S}\mathrm{et}$ defining for every signature Σ the set $\mathsf{Sen}^{\mathcal{I}}(\Sigma)$ of Σ-*sentences*, and for every signature morphism $\varphi\colon \Sigma \to \Sigma'$ the *sentence translation map* $\mathsf{Sen}^{\mathcal{I}}(\varphi)\colon \mathsf{Sen}^{\mathcal{I}}(\Sigma) \to \mathsf{Sen}^{\mathcal{I}}(\Sigma')$,
- a *model functor* $\mathsf{Mod}^{\mathcal{I}}\colon (\mathbb{S}\mathrm{ig}^{\mathcal{I}})^{op} \to \mathbb{C}\mathrm{at}$ defining for every signature Σ the category $\mathsf{Mod}^{\mathcal{I}}(\Sigma)$ of Σ-*models* and Σ-*homomorphisms*, and for every morphism $\varphi\colon \Sigma \to \Sigma'$, the *reduct functor* $\mathsf{Mod}^{\mathcal{I}}(\varphi)\colon \mathsf{Mod}^{\mathcal{I}}(\Sigma') \to \mathsf{Mod}^{\mathcal{I}}(\Sigma)$,
- a family of *satisfaction* relations $\models^{\mathcal{I}}_{\Sigma} \subseteq |\mathsf{Mod}^{\mathcal{I}}(\Sigma)| \times \mathsf{Sen}^{\mathcal{I}}(\Sigma)$, indexed by signatures,

such that the following *satisfaction condition* holds:

$$M' \models^{\mathcal{I}}_{\Sigma'} \mathsf{Sen}^{\mathcal{I}}(\varphi)(\rho) \quad \text{if and only if} \quad \mathsf{Mod}^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho,$$

for every signature morphism $\varphi\colon \Sigma \to \Sigma'$, Σ'-model M' and Σ-sentence ρ.

2 Asynchronous Relational Networks

The first step towards the formulation of the logical framework for service orchestrations is the introduction of a parameterised construction of a category $\langle \mathbb{S}\mathrm{ig}, L\rangle$-$\mathbb{ARN}$ of asynchronous relational networks. In the subsequent sections of the present paper this will serve as foundation for both the syntactic and the semantic dimensions of the proposed logical system.

Throughout this section we will consider

- a fixed category $\mathbb{S}\mathrm{ig}$ (of *signatures* and *signature morphisms*) and
- a *behavioural labelling functor* $L\colon |\mathbb{S}\mathrm{ig}| \to \mathbb{C}\mathrm{at}$ assigning to every signature $\Sigma \in |\mathbb{S}\mathrm{ig}|$ a category $L(\Sigma)$ of *behavioural* Σ-*labels*.

For presentation purposes, we will assume $\mathbb{S}\mathrm{ig}$ to be the category of signatures of linear temporal logic, and $L(\Sigma)$ to be the category of Σ-presentations, i.e. of sets of Σ-sentences.

The asynchronous relational networks that we examine here follow closely the formalisation from [3], and are similar to those defined in [6] except that we rely on hypergraphs [5, 2] instead of graphs, and on abstract labels instead of signatures of linear temporal logic and sets of traces. Just as in [6], the proposed concepts reflect the intuitive representation of service components as networks of processes that interact asynchronously by exchanging messages through communication channels. However, since the role of messages (organised into structures called ports) is merely to provide a convenient description of certain signatures of linear temporal logic, we choose not to introduce them explicitly, but to refer directly to the signatures they designate, similarly to [9].

Processes are defined by sets of interaction-points labelled with port signatures and by behavioural labels that correspond to the way the processes operate.

▶ **Definition 1** (Process). A $\langle \mathbb{S}\mathrm{ig}, L\rangle$-*process* $\langle X, (\Sigma_x \xrightarrow{\iota_x} \Sigma)_{x \in X}, \Lambda\rangle$ consists of a set X of *interaction-points*, each point $x \in X$ being labelled with a *port signature* $\Sigma_x \in |\mathbb{S}\mathrm{ig}|$, a *process signature* $\Sigma \in |\mathbb{S}\mathrm{ig}|$ such that $(\Sigma_x \xrightarrow{\iota_x} \Sigma)_{x \in X}$ is a coproduct in $\mathbb{S}\mathrm{ig}$, and a label $\Lambda \in |L(\Sigma)|$.

In the actual situations the port signatures Σ_x are sets of actions that match either the *publication* $m!$ of an outgoing message m or the *delivery* $m¡$ of an incoming message m. The process signature Σ is given by the disjoint union $\biguplus_{x \in X} \Sigma_x = \{x.a \mid x \in X, a \in \Sigma_x\}$, while the injections ι_x are the obvious prefix maps $x.\_\colon \Sigma_x \to \Sigma$. The behaviour of the process is specified through a set Λ of sentences of linear temporal logic over Σ.

▶ **Example 2.** In Figure 1a we depict a process JourneyPlanner that provides directions from a source to a target location. The process interacts with the environment through two interaction-points: JP_1 and JP_2. The first one is used for communicating with potential

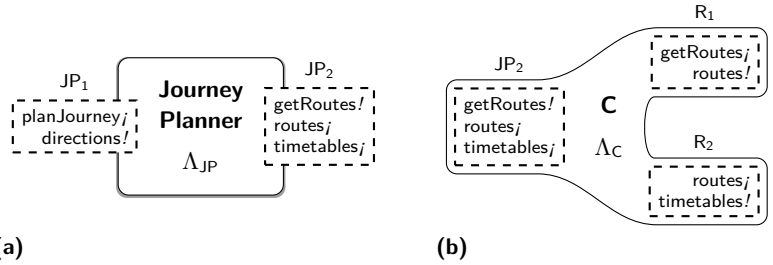


Figure 1 The ARN JourneyPlanner: (a) the process JourneyPlanner, (b) the connection C.

client processes – the request for directions (including the source and the target locations) is encoded into the delivery action planJourney¡, while the response is represented by the publication action directions!. The second point defines actions that correspond to the interaction between JourneyPlanner and other necessary processes – the publication action getRoutes! can be seen as a query for all possible routes between the specified source and target locations, while routes¡ and timetables¡ define the delivery of the result of the query and of the timetables of the available transport services for the selected routes.

The behaviour of the process JourneyPlanner can be described as follows:

– Whenever it receives a request planJourney¡ it immediately initiates the search of available routes through the publication action getRoutes!.

$$\Box\,(\mathsf{planJourney}_{¡} \supset \bigcirc\,\mathsf{getRoutes}!)$$

– Once it receives both the routes and the corresponding timetables it compiles the directions and replies to the client.

$$\Box\,\Big(\mathsf{getRoutes}! \supset \big(\mathsf{routes}_{¡}\,\mathcal{R}\,(\mathsf{routes}_{¡} \supset (\mathsf{timetables}_{¡}\,\mathcal{R}\,(\mathsf{timetables}_{¡} \supset \Diamond\,\mathsf{directions}!))) \vee \mathsf{timetables}_{¡}\,\mathcal{R}\,(\mathsf{timetables}_{¡} \supset (\mathsf{routes}_{¡}\,\mathcal{R}\,(\mathsf{routes}_{¡} \supset \Diamond\,\mathsf{directions}!))))\big)\Big)$$

Processes communicate by transmitting messages through channels. As in [1, 6], channels are bidirectional, in the sense that they may transmit both incoming and outgoing messages.

▶ **Definition 3** (Channel). A $\langle \mathbb{S}\mathrm{ig}, L\rangle$-*channel* $\langle \Sigma, \Lambda\rangle$ consists of a *channel signature* $\Sigma \in |\mathbb{S}\mathrm{ig}|$ and a behavioural label $\Lambda \in |L(\Sigma)|$.

Note that channels do not provide any information about the interacting entities, but only on how the communication is realised. In order to enable the communication between given processes, channels need to be attached to their interaction-points, thus forming connections.

▶ **Definition 4** (Connection). A $\langle \mathbb{S}\mathrm{ig}, L\rangle$-*connection* $\langle \Sigma, \Lambda, (\Sigma \xleftarrow{\iota_x} \Sigma'_x \xrightarrow{\mu_x} \Sigma_x)_{x\in X}\rangle$ between the port signatures $(\Sigma_x)_{x\in X}$, where X is a set of *interaction-points*, consists of a channel $\langle \Sigma, \Lambda\rangle$ and a family of *attachment spans* $(\Sigma \xleftarrow{\iota_x} \Sigma'_x \xrightarrow{\mu_x} \Sigma_x)_{x\in X}$ in $\mathbb{S}\mathrm{ig}$.

The channel signature Σ is given in general by a set of both publication and delivery actions $m!$ and $m_¡$ determined by messages m. As in the case of processes, the behaviour of the channel is specified through a set of sentences of linear temporal logic over Σ. The maps ι_x and μ_x are considered to be set-theoretic inclusions and polarity-preserving injections, respectively, i.e. injective functions μ_x with the property that $\mu_x(a)$ is a publication action of Σ_x if and only if a is a publication action of Σ, and for this reason they are often presented as

partial maps $\mu_x \colon \Sigma \rightharpoonup \Sigma_x$; in addition, the signatures $(\Sigma'_x)_{x \in X}$ are usually chosen such that for any point $x \in X$, $m! \in \Sigma'_x$ ($m¿ \in \Sigma'_x$) if and only if $m¿ \in \Sigma'_y$ ($m! \in \Sigma'_y$) for some $y \in X \setminus \{x\}$. This condition ensures an appropriate pairing of messages: every published message of Σ_x, for $x \in X$, is paired with a delivered message of Σ_y, for some $y \in X \setminus \{x\}$, and vice versa.

▶ **Example 5.** In order to illustrate how the process JourneyPlanner can interact with other processes, we consider the connection C depicted in Figure 1b that moderates the flow of messages between the port signature named JP_2 and two other port signatures, R_1 and R_2.

The underlying channel of C is given by the set of actions $\Sigma = \{g!, g¿, r!, r¿, t!, t¿\}$ together with the set of sentences of linear temporal logic

$$\Lambda_{\mathsf{C}} = \left\{ \Box \left(m! \to \bigcirc m¿ \right) \mid m \in \{g, r, t\} \right\}$$

that specifies the delivery of all published messages without any delay. It is attached to the port signatures JP_2, R_1 and R_2 through the partial injections μ_{JP_2}, μ_{R_1} and μ_{R_2} given by

- $\mu_{\mathsf{JP}_2} = \{g! \mapsto \mathsf{getRoutes}!, r¿ \mapsto \mathsf{routes}¿, t¿ \mapsto \mathsf{timetables}¿\}$,
- $\mu_{\mathsf{R}_1} = \{g¿ \mapsto \mathsf{getRoutes}¿, r! \mapsto \mathsf{routes}!\}$ and
- $\mu_{\mathsf{R}_2} = \{r¿ \mapsto \mathsf{routes}¿, t! \mapsto \mathsf{timetables}!\}$.

Note that the actual senders and receivers of messages are specified through the attachment injections. For example, the message g is delivered only to the port signature R_1 (because μ_{R_2} is not defined on $g¿$), while r is simultaneously delivered to both JP_2 and R_2.

We can now define asynchronous networks of processes as hypergraphs having vertices labelled with port signatures and hyperedges labelled with processes and connections.

▶ **Definition 6** (Hypergraph)**.** An *(edge-labelled) hypergraph* $\langle X, E, \gamma \rangle$ consists of a set X of *vertices* or *nodes*, a set E of *hyperedges* disjoint from X, and an *incidence map* $\gamma \colon E \to \mathcal{P}(X)$ defining for every hyperedge $e \in E$ a non-empty set $\gamma_e \subseteq X$ of vertices it is incident with.

A hypergraph $\langle X, E, \gamma \rangle$ is said to be *edge-bipartite* if E is partitioned into two subsets F and G such that no adjacent hyperedges belong to the same partition, i.e. for every two hyperedges $e_1, e_2 \in E$ such that $\gamma_{e_1} \cap \gamma_{e_2} \neq \emptyset$, either $e_1 \in F$ and $e_2 \in G$, or $e_1 \in G$ and $e_2 \in F$.

▶ **Definition 7** (Asynchronous Relational Network – ARN)**.** A $\langle \mathbb{S}\mathrm{ig}, L \rangle$*-asynchronous relational network* $A = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda \rangle$ consists of an edge-bipartite hypergraph $\langle X, P, C, \gamma \rangle$ of *points* $x \in X$, *computation hyperedges* $p \in P$, *communication hyperedges* $c \in C$, together with

- a *port signature* $\Sigma_x \in |\mathbb{S}\mathrm{ig}|$ for every point $x \in X$,
- a process $\left\langle \gamma_p, \left(\Sigma_x \xrightarrow{\iota^p_x} \Sigma_p \right)_{x \in \gamma_p}, \Lambda_p \right\rangle$ for every hyperedge $p \in P$, and
- a connection $\left\langle \Sigma_c, \Lambda_c, \left(\Sigma_c \xleftarrow{\iota^c_x} \Sigma^c_x \xrightarrow{\mu^c_x} \Sigma_x \right)_{x \in \gamma_c} \right\rangle$ for every hyperedge $c \in C$.

▶ **Example 8.** By putting together the process and the connection presented in Examples 2 and 5, we obtain the ARN JourneyPlanner depicted in Figure 1. Its underlying hypergraph consists of the points JP_1, JP_2, R_1 and R_2, the computation hyperedge JP, the communication hyperedge C, and the incidence map γ given by $\gamma_{\mathsf{JP}} = \{\mathsf{JP}_1, \mathsf{JP}_2\}$ and $\gamma_{\mathsf{C}} = \{\mathsf{JP}_2, \mathsf{R}_1, \mathsf{R}_2\}$.

An *interaction-point* of a $\langle \mathbb{S}\mathrm{ig}, L \rangle$-ARN A is a point of A that is not bound to both computation and communication hyperedges. We distinguish between requires-points and provides-points, as follows.

▶ **Definition 9** (Requires and Provides-point)**.** A *requires-point* of $\langle \mathbb{S}\mathrm{ig}, L \rangle$-ARN A is a point of A that is incident only with a communication hyperedge. Similarly, a *provides-point* of A is a point incident only with a computation hyperedge.

Morphisms of $\langle \mathbb{S}\mathrm{ig}, L\rangle$-ARNs can be defined as injective homomorphisms between their underlying hypergraphs that preserve all labels, except those associated with requires-points.

▶ **Definition 10** (Homomorphism of Hypergraphs). A *homomorphism* h between hypergraphs $\langle X, E, \gamma\rangle$ and $\langle X', E', \gamma'\rangle$ consists of functions $h^v\colon X \to X'$ and $h^e\colon E \to E'$, usually denoted simply by h, such that for any $x \in X$ and $e \in E$, $x \in \gamma_e$ if and only if $h^v(x) \in \gamma'_{h^e(e)}$.

▶ **Definition 11** (Morphism of ARNs). Given two $\langle \mathbb{S}\mathrm{ig}, L\rangle$-ARNs $A = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda\rangle$ and $A' = \langle X', P', C', \gamma', \Sigma', \iota', \mu', \Lambda'\rangle$, a *morphism* $\varphi\colon A \to A'$ consists of

- an injective homomorphism $\varphi\colon \langle X, P, C, \gamma\rangle \to \langle X', P', C', \gamma'\rangle$ between the underlying hypergraphs of A and A' such that $\varphi(P) \subseteq P'$ and $\varphi(C) \subseteq C'$, and
- a family φ^{pt} of signature morphisms $\varphi^{pt}_x\colon \Sigma_x \to \Sigma'_{\varphi(x)}$, for $x \in X$,

such that

- for every non-requires-point $x \in X$, $\varphi^{pt}_x = 1_{\Sigma_x}$,
- for every hyperedge $p \in P$, $\Sigma_p = \Sigma'_{\varphi(p)}$, $\Lambda_p = \Lambda'_{\varphi(p)}$, and $\iota^p_x = (\iota')^{\varphi(p)}_{\varphi(x)}$ for any point $x \in \gamma_p$,
- for every hyperedge $c \in C$, $\Sigma_c = \Sigma'_{\varphi(c)}$, $\Lambda_c = \Lambda'_{\varphi(c)}$ and the following diagram is well-defined and commutative, for any point $x \in \gamma_c$.

$$\begin{array}{ccccc}
\Sigma_c & \xleftarrow{\iota^c_x} & \Sigma^c_x & \xrightarrow{\mu^c_x} & \Sigma_x \\
\downarrow{\scriptstyle 1_{\Sigma_c}} & & \downarrow{\scriptstyle 1_{\Sigma^c_x}} & & \downarrow{\scriptstyle \varphi^{pt}_x} \\
\Sigma'_{\varphi(c)} & \xleftarrow[(\iota')^{\varphi(c)}_{\varphi(x)}]{} & (\Sigma')^{\varphi(c)}_{\varphi(x)} & \xrightarrow[(\mu')^{\varphi(c)}_{\varphi(x)}]{} & \Sigma'_{\varphi(x)}
\end{array}$$

▶ **Proposition 12.** *The morphisms of $\langle \mathbb{S}\mathrm{ig}, L\rangle$-ARNs form a category denoted $\langle \mathbb{S}\mathrm{ig}, L\rangle$-$\mathbb{ARN}$.*

3 An institution of ARNs

We now turn our attention to the main contribution of the paper: the presentation of a logical framework for service orchestrations in an institutional setting. Similarly to the construction detailed in the previous section, the concepts discussed here are parameterised over an arbitrary logical system that satisfies a number of additional properties. More precisely, we consider a fixed institution $\mathcal{I} = \langle \mathbb{S}\mathrm{ig}^{\mathcal{I}}, \mathsf{Sen}^{\mathcal{I}}, \mathsf{Mod}^{\mathcal{I}}, \models^{\mathcal{I}}\rangle$ such that

- the category $\mathbb{S}\mathrm{ig}^{\mathcal{I}}$ of $\mathcal{I}$-signatures is cocomplete,
- there exist cofree models along any signature morphism, i.e. the reduct functor $\mathsf{Mod}^{\mathcal{I}}(\varphi)$ of any signature morphism φ admits a right adjoint, and
- the category of models $\mathsf{Mod}^{\mathcal{I}}(\Sigma)$ of any signature Σ has products.

An example of such institution is MA-LTL – a variant of linear temporal logic whose models are not traces, but Muller automata [11], and in which an automaton satisfies a sentence if and only if every trace accepted by the automaton satisfies the considered sentence. MA-LTL was originally proposed in [3] as an alternative to conventional linear temporal logic with the aim of capturing a more operational notion of service orchestration.

3.1 Signatures

We start by defining the category $\mathbb{S}\mathrm{pec}$-$\mathbb{ARN}_{\mathcal{I}}$ of signatures and signature morphisms of our institution, which we denote $\mathrm{SOC}_{\mathcal{I}}$. These are asynchronous relational networks determined

by the category $\mathbb{S}\mathrm{ig}^{\mathcal{I}}$ of signatures and by the labelling functor $\mathsf{Spec}^{\mathcal{I}}\colon |\mathbb{S}\mathrm{ig}^{\mathcal{I}}| \to \mathbb{C}\mathrm{at}$ that assigns to every $\mathcal{I}$-signature Σ the preorder category of Σ-presentations.

$$\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}} = \langle \mathbb{S}\mathrm{ig}^{\mathcal{I}}, \mathsf{Spec}^{\mathcal{I}} \rangle\text{-}\mathbb{ARN}$$

3.2 Sentences and Sentence Translations

The sentences of $\underline{\mathrm{SOC}}_{\mathcal{I}}$ express properties about the points of the considered ARNs.

▶ **Definition 13** (Sentence). For any network $A \in |\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}|$, i.e. for any $\underline{\mathrm{SOC}}_{\mathcal{I}}$-signature A, the set $\mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(A)$ of *(atomic) A-sentences* is defined as the set of pairs $\langle x, \rho \rangle$, usually denoted $@_x\, \rho$, where x is a point of A and ρ is an $\mathcal{I}$-sentence over Σ_x.

The *translation* of sentences is straightforward: for every morphism $\varphi\colon A \to A'$ in $\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}$, the map $\mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)\colon \mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(A) \to \mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(A')$ is given by

$$\mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)(@_x\, \rho) = @_{\varphi(x)}\, \mathsf{Sen}^{\mathcal{I}}(\varphi^{pt}_x)(\rho)$$

for any point x of A and any $\mathcal{I}$-sentence ρ over the signature of x.

▶ **Proposition 14.** $\mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}$ *is a functor* $\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}} \to \mathbb{S}\mathrm{et}$.

3.3 Models and Model Reductions

The model functor of our institution assigns appropriate models of the underlying institution to the computation and communication hyperedges of the considered ARNs, and ground networks to their requires-points. Formally, we first define

- $\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}$ as the category $\langle \mathbb{S}\mathrm{ig}^{\mathcal{I}}, |\mathsf{Mod}^{\mathcal{I}}| \rangle\text{-}\mathbb{ARN}$,
- $\mathbb{M}\text{od-}\mathbb{ARN}^{A}_{\mathcal{I}}$, for any ARN $A = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda^{spec} \rangle \in |\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}|$, as the discrete subcategory of $\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}$ given by networks $\alpha = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda^{mod} \rangle$ such that $\Lambda^{mod}_e \models^{\mathcal{I}}_{\Sigma_e} \Lambda^{spec}_e$ for every computation or communication hyperedge e of A, and
- $\mathbb{G}\mathbb{ARN}_{\mathcal{I}}$ as the full subcategory of $\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}$ determined by *ground* networks, i.e. by networks with no requires-points.

▶ **Definition 15** (Model). For any ARN $A \in |\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}|$, the category $\mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}(A)$ of *A-models* or *A-interpretations* is defined as the comma category $\mathbb{M}\text{od-}\mathbb{ARN}^{A}_{\mathcal{I}} / \mathbb{G}\mathbb{ARN}_{\mathcal{I}}$.

It follows that A-interpretations are morphisms of ARNs $\nu\colon \alpha \to \beta$ such that $\alpha \in |\mathbb{M}\text{od-}\mathbb{ARN}^{A}_{\mathcal{I}}|$ and $\beta \in |\mathbb{G}\mathbb{ARN}_{\mathcal{I}}|$, which can also be seen as collections of ground networks that are designated to the requires-points of α. In order to explain this in more detail let us introduce the following notions of dependency and ARN defined by a point.

▶ **Definition 16** (Dependency). Let x and y be points of an ARN α. The point x is said to be *dependent* on y if there exists a path from x to y that begins with a computation hyperedge, i.e. if there exists an alternating sequence $x\, e_1\, x_1 \cdots e_n\, y$ of (distinct) points and hyperedges such that $x \in \gamma_{e_1}$, $y \in \gamma_{e_n}$, $x_i \in \gamma_{e_i} \cap \gamma_{e_{i+1}}$ for any $1 \leq i < n$, and $e_1 \in P$.

▶ **Definition 17** (ARN Defined by a Point). The *sub-ARN defined by a point* x of an ARN α is the full sub-ARN α_x of α determined by x and the points on which x is dependent.

One can now see that any interpretation $\nu\colon \alpha \to \beta$ of an ARN $A \in |\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}|$ assigns to each requires-point x of α the ground sub-ARN $\beta_{\nu(x)}$ of β defined by $\nu(x)$.

With respect to model reducts, one can easily see that every ARN morphism $\varphi\colon A \to A'$ in $\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}$ and every network $\alpha' \in |\mathbb{M}\text{od-}\mathbb{ARN}^{A'}_{\mathcal{I}}|$ determine a (unique) morphism of ARNs

$\varphi\colon \alpha \to \alpha'$ in $\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}$ such that $\alpha \in |\mathbb{M}\text{od-}\mathbb{ARN}^{A}_{\mathcal{I}}|$. This observation allows us to define the reduction of interpretations as the left composition with the considered ARN morphism. Hence, for every ARN morphism $\varphi\colon A \to A'$, $\mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)$ is given by $\mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)(\nu') = \varphi;\nu'$ for A'-interpretations ν' and $\mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)(\zeta') = \zeta'$ for A'-interpretation homomorphisms ζ'.

▶ **Proposition 18.** $\mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}$ *is a contravariant functor* $\mathbb{S}\text{pec-}\mathbb{ARN}^{op}_{\mathcal{I}} \to \mathbb{C}\text{at}$.

3.4 The Satisfaction Relation

The evaluation of $\underline{\mathrm{SOC}}_{\mathcal{I}}$ sentences with respect to the interpretations relies on the concepts of diagram of a network and of model defined by a point, whose purpose is to describe the observable behaviour of a ground network through one of its points.

▶ Fact 19 (Diagram of an ARN). Every ARN $\alpha = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda^{mod}\rangle \in |\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}|$ defines a diagram $D_\alpha\colon \mathbb{J}_\alpha \to \mathbb{S}\text{ig}^{\mathcal{I}}$ as follows:

– $\mathbb{J}_\alpha$ is the free preorder category given by the set of objects

$$X \cup P \cup C \cup \{\langle c, x, \alpha\rangle \mid c \in C, x \in \gamma_c\}$$

and the arrows

- $\{x \to p \mid p \in P, x \in \gamma_p\}$ for computation hyperedges, and
- $\{c \leftarrow \langle c, x, \alpha\rangle \to x \mid c \in C, x \in \gamma_c\}$ for communication hyperedges;

– D_α is the functor that provides the signatures of ports, processes and channels, together with the appropriate mappings between them.

Since $\mathbb{S}\text{ig}^{\mathcal{I}}$ is cocomplete, we can define the signature of a network based on its diagram.

▶ **Definition 20** (Signature of an ARN)**.** The signature of an ARN $\alpha \in |\mathbb{M}\text{od-}\mathbb{ARN}_{\mathcal{I}}|$ is the colimiting cocone $\xi\colon D_\alpha \Rightarrow \Sigma_\alpha$ of the diagram D_α.

The most important construction that allows us to define the satisfaction relation is the one that defines the observed behaviour of a (ground) network at one of its points.

▶ **Definition 21** (Model Defined by a Point)**.** Let x be a point of a ground ARN $\beta \in |\mathbb{G}\mathbb{ARN}_{\mathcal{I}}|$. The *observed model* Λ^{mod}_x at x is given by the reduct $\mathsf{Mod}^{\mathcal{I}}(\xi_x)(\Lambda^{mod}_{\beta_x})$, where

– $\beta_x = \langle X, P, C, \gamma, \Sigma, \iota, \mu, \Lambda^{mod}\rangle$ is the sub-ARN of β defined by x,
– $\xi\colon D_{\beta_x} \Rightarrow \Sigma_{\beta_x}$ is the signature of β_x,
– $\Lambda^{mod}_{\beta_x}$ is the product $\prod_{e \in P \cup C} \Lambda^{mod}_{\beta_x, e}$, and
– $\Lambda^{mod}_{\beta_x, e}$ is the cofree expansion of Λ^{mod}_e along ξ_e, for any hyperedge $e \in P \cup C$.

We now have all the necessary concepts for defining the satisfaction of $\underline{\mathrm{SOC}}_{\mathcal{I}}$ sentences by interpretations. Let us thus consider an ARN $A \in |\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}|$, an A-interpretation $\nu\colon \alpha \to \beta$ and an A-sentence $@_x\,\rho$. Then

$$\nu \models^{\mathrm{SOC}}_{\mathcal{I},A} @_x\,\rho \quad \text{if and only if} \quad \mathsf{Mod}^{\mathcal{I}}(\nu^{pt}_x)\big(\Lambda^{mod}_{\nu(x)}\big) \models^{\mathcal{I}}_{\Sigma_x} \rho,$$

where $\Lambda^{mod}_{\nu(x)}$ is the observed model at $\nu(x)$ in β.

The construction of the institution of ARNs is completed by the following result, which states that satisfaction is invariant with respect to changes of ARNs.

▶ **Proposition 22.** *For every ARN morphism* $\varphi\colon A \to A'$ *in* $\mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}$*, any* A'*-interpretation* ν' *and any* A*-sentence* $@_x\,\rho$*,*

$$\nu' \models^{\mathrm{SOC}}_{\mathcal{I},A'} \mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)(@_x\,\rho) \quad \text{if and only if} \quad \mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}(\varphi)(\nu') \models^{\mathrm{SOC}}_{\mathcal{I},A} @_x\,\rho.$$

▶ **Corollary 23.** $\underline{\mathrm{SOC}}_{\mathcal{I}} = \big\langle \mathbb{S}\text{pec-}\mathbb{ARN}_{\mathcal{I}}, \mathsf{Sen}^{\mathrm{SOC}}_{\mathcal{I}}, \mathsf{Mod}^{\mathrm{SOC}}_{\mathcal{I}}, \models^{\mathrm{SOC}}_{\mathcal{I}}\big\rangle$ *is an institution.*

4 Conclusions

In this paper we proposed a logical framework of networks of processes that can be used in combination with concepts and results specific to the logic-programming paradigm [3] to offer an integrated semantics for the static and dynamic aspects of service-oriented computing. We distinguish between specifications of services and their models, which are orchestrations of components that rely upon externally provided services. The resulting institution is parameterised over an arbitrary logical system such that (a) its category of signatures is cocomplete, (b) there exist cofree models along any signature morphism, (c) the category of models of any signature has products. This level of generality encourages us to further investigate the service-oriented computing paradigm over a variety of logics. An issue to be pursued towards this heterogeneous setting is the way in which the change of the underlying logics induces appropriate translations between the corresponding institutions of ARNs.

Acknowledgements. The author would like to thank José Fiadeiro for many useful discussions and feedback on the logic-programming semantics of services. This research has been supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439.

References

1 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

2 Roberto Bruni, Fabio Gadducci, and Alberto Lluch-Lafuente. A graph syntax for processes and services. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods*, Lecture Notes in Computer Science, pages 46–60. Springer, 2009.

3 Ionuţ Ţuţu and José L. Fiadeiro. A logic-programming semantics of services. In Reiko Heckel and Stefan Milius, editors, *Conference on Algebra and Coalgebra in Computer Science*, Lecture Notes in Computer Science, pages 299–313. Springer, 2013.

4 Răzvan Diaconescu. *Institution-independent model theory*. Studies in Universal Logic. Birkhäuser, 2008.

5 Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In Frank de Boer, Marcello Bonsangue, Susanne Graf, and Willem de Roever, editors, *Formal Methods for Components and Objects*, Lecture Notes in Computer Science, pages 22–43. Springer, 2005.

6 José L. Fiadeiro and Antónia Lopes. An interface theory for service-oriented design. In Dimitra Giannakopoulou and Fernando Orejas, editors, *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, pages 18–33. Springer, 2011.

7 José L. Fiadeiro and Antónia Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and Systems Modeling*, pages 1–19, 2012.

8 José L. Fiadeiro, Antónia Lopes, and Laura Bocchi. Algebraic semantics of service component modules. In José L. Fiadeiro and Pierre-Yves Schobbens, editors, *Workshop on Algebraic Development Techniques*, Lecture Notes in Computer Science, pages 37–55. Springer, 2006.

9 José L. Fiadeiro, Antónia Lopes, and Laura Bocchi. An abstract model of service discovery and binding. *Formal Aspects of Computing*, 23(4):433–463, 2011.

10 Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

11 Dominique Perrin and Jean Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier Science, 2004.

Refactoring Boundary

Tim Wood[1] and Sophia Drossopoulou[2]

1 **Imperial College London** t.wood12@imperial.ac.uk
2 **Imperial College London** s.drossopoulou@imperial.ac.uk

Abstract

We argue that the limit of the propagation of the heap effects of a source code modification is determined by the aliasing structure of method parameters in a trace of the method calls that cross a boundary which partitions the heap. Further, that this aliasing structure is sufficient to uniquely determine the state of the part of the heap which has not been affected. And we give a definition of what it means for a part of the heap to be unaffected by a source code modification. This can be used to determine the correctness of a refactoring.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Refactoring, Object Oriented

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.119

1 Introduction

A refactoring is a transformation applied to the source code of a program that preserves the meaning of the program. Most refactorings however do affect the internal behaviour of a program, for example, new classes and methods may be introduced, conditionals may be replaced with polymorphism and classes may be in-lined. Determining if a particular transformation is indeed meaning preserving is a challenge.

According to our proposal, the heap is partitioned into an affected part and an unaffected part. Moreover we give a sufficient condition that determines that the effects of the code modification do not spread from the affected part into the unaffected part — which we call a *bounded* modification.

Motivation: A programmer modifying a program wants to avoid breaking existing functionality, even functionality that they may not be aware of. A programmer wants to convince other programmers to accept their patch, and wants to show that their change is safe. A programmer wants to precisely characterise the difference between two versions of a program.

Contributions: We argue that a good way to characterise the difference between two versions of a program is to partition the heap into an affected part and an unaffected part, and require that executions correspond in the unaffected part. We propose a precise definition of this novel property. We propose a novel sufficient condition for such a correspondence of heaps — that there will be a correspondence in the unaffected part whenever the aliasing structure of stack frames witnessed at the heap partition boundary is isomorphic between executions of the two versions. We anticipate that this sufficient condition will provide a basis for automated tools that can check if a modification is bounded

Our proposal gives a novel general definition of refactoring correctness in terms of the heap effect of the modification. Moreover it offers a wider definition of refactoring correctness than other approaches, depending on what unaffected partition is picked. For example, if all I/O occurs in the unaffected partition then our approach is similar to traditional definitions

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jones, Nicholas Ng; pp. 119–127

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of refactoring correctness, but other choices of partition are possible that allow differences in program I/O. This notion of correctness does not require any additional specification of the program, nor does it require the programmer to limit themselves to previously known-correct refactorings with established pre-conditions.

Related Work: Typically the correctness of refactorings is defined as I/O equivalence and is checked by ad-hoc pre-condition checking[10][1] or by composing smaller refactorings[12]. State based encapsulation provides a method for reasoning about the equivalence of classes[9]. Program slicing can be used to establish the correctness of some statement reordering refactorings[2]. Graph transformations have been suggested as a means to reason about refactorings in general[11]. Regression verification uses bounded model checking to verify the equivalence of some loop and recursion free programs[3]. Program verification tools in conjunction with automated theorem provers can be used to check some programs for equivalence[4][6], or library versions for backward compatibility[13]. Semantics aware trace analysis[5] and BCT[7] use traces captured at runtime to attempt to isolate the causes of regression failures. Guru[8] uses sequences of messages sends to define equivalence of differently factored method implementations in an object-oriented system.

Structure: In **section 2** we give a motivating example, and describe what it means for the effect of a refactoring to be bounded. In **section 3** we give a sufficient condition for determining if a part of the heap is unaffected between executions of two versions of the program. Then in **section 4** we describe exactly how partitioning and trace comparison works. In **section 5** we give several interesting properties of bounded modifications, and in **section 6** sketch a proof of those properties.

2 Bounded Effect

An object oriented program p is modified to produce a program q. The heap is partitioned into two parts Φ^m and Φ^u. We say that Φ^m *bounds the modification* if for any execution in p there is a corresponding execution in q such that the heaps are equivalent within Φ^u. In other words, the heap effect of a bounded modification does not spread beyond Φ^m.

```
1  class FiFo {
2   Node f;
3   void add(final Object o) {
4    if(f == null) { f=new Node(o); }
5    else { f.add(o); }}
6   Object remove() {
7    Object r=f.value(); f=f.next(); return
         r;}}
8
9  class Node {
10  Object o; Node n;
11  Node(Object o) { this.o=o; }
12  Node next() { return n; }
13  Object value() { return o; }
14  void add(final Object o) {
15   if(n == null) { n=new Node(o); }
16   else { n.add(o); }}}
```

Listing 1 Program p — fifo queue with recursive add.

```
1  class FiFo {
2   Node f, l;
3   void add(final Object o) {
4    if(f == null) { l=f=new Node(o); }
5    else { l=l.add(o); }}
6   Object remove() {
7    Object r=f.value(); f=f.next(); return
         r;}}
8
9  class Node {
10  Object o; Node n;
11  Node(Object o) { this.o=o; }
12  Node next() { return n; }
13  Object value() { return o; }
14   Node add(Object o) {
15   return n=new Node(o); }}
16
```

Listing 2 Program q — fifo queue with last element pointer. The modified parts are highlighted.

```
1  class Test { Fifo fi = new Fifo(); Object o1 = new Object(), o2 = new Object();
2   void test() { fi.add(o1); fi.add(o2); fi.remove(); fi.remove();}}
```

Listing 3 Test program for the FiFo queue.

We shall clarify the meaning of execution, correspondence, partitioning and equivalence in terms of the following example. Listing 1 shows part of a larger program `p`. The code shown is a fifo queue consisting of two classes `FiFo` and `Node`. The code is modified, as in listing 2, to produce a new version `q` where the implementation and representation of the queue has changed. Listing 3 shows a test program that could be run with the code from either Listing 1 or Listing 2. The only values in our language are addresses, which can be compared for equality but are otherwise opaque, and all fields are private.

Consider Figure 1 which shows the stack and heap when an execution of the test program in Listing 3 with `p` is just about to return from the second call to `add` on line 2 of Listing 3. In this figure Φ^{m} is chosen to include only objects of the `FiFo` and `Node` classes.

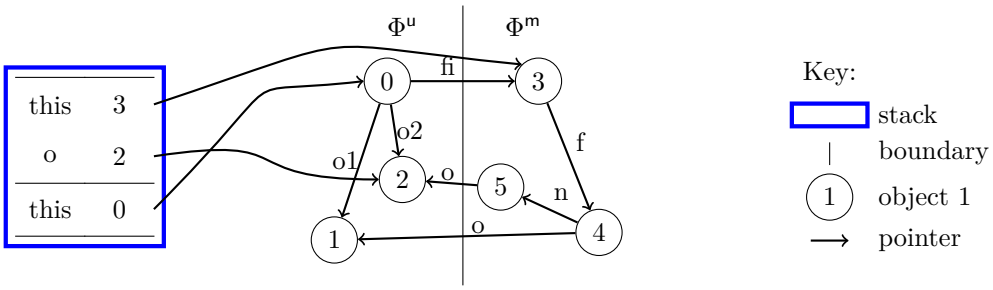


Figure 1 Snapshot of an execution of `p` (Listing 3 and Listing 1). The heap and top two stack frames are shown. The heap is partitioned so that objects at addresses 0,1,2 are in Φ^{u}, and objects 3,4,5 are in Φ^{m}. The top stack frame has a `this` pointer into the Φ^{m} partition, the stack frame below it has a `this` pointer into the Φ^{u} partition.

In Figure 2 we show the stack and heap at a corresponding point in the execution of the test program in Listing 3 with `q`. Notice that when compared to Figure 1 this figure has an additional pointer between objects 10 and 12, this is due to the field `l` which has been added in `q` as shown on line 2 of Listing 2.

We notice that the heap in Figure 1, which we will call h_1, and the heap in Figure 2, which we will call h_2, are *isomorphic* when only the objects within Φ^{u} are considered. We say that h_1 is equivalent to h_2 wrt. Φ^{u}, and in our notation we write this as $h_1 \approx_{\Phi^{\mathsf{u}}} h_2$. In this case, object 0 corresponds with object 7, 1 with 8, and 2 with 9.

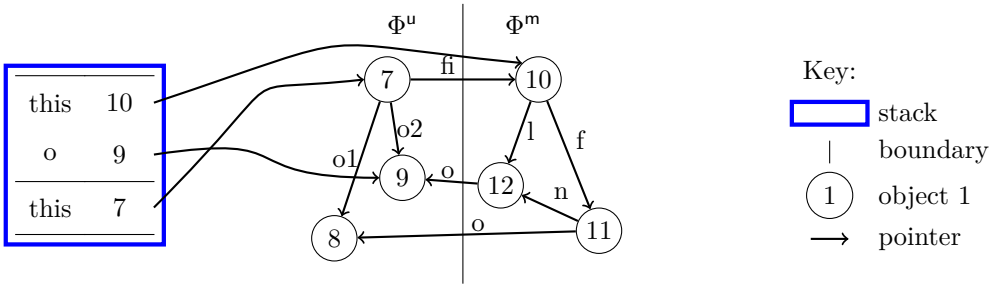


Figure 2 Snapshot of an execution of `q` (Listing 3 and Listing 2). Notice that there is an extra pointer between objects 10 and 12 when compared to Figure 1.

Definition 1 gives the formal meaning of bounded, where we use $\gamma, \gamma_p, \gamma_q$ to range over runtime configurations, and $\gamma \longrightarrow\!\!\!\!\!\rightarrow_p \gamma'$ to mean the multi-step execution of program p from configuration γ to configuration γ'. We write $\mathsf{heap}(\gamma)$ to mean the heap of configuration γ. The judgement $\mathsf{init}(\gamma)$ holds when γ is an initial state, which means it has an empty stack and an expression consisting of a call to the program's main method.

▶ **Definition 1** (bounded). When Φ^{m} *bounds the effect*, for every sequence of states reachable in p there is a sequence of states with equivalent heaps wrt. Φ^{u} reachable in q.

$$\forall \gamma, \gamma_p, \gamma'_p : \mathsf{init}(\gamma) \ \wedge \ \gamma \longrightarrow\!\!\!\!\!\rightarrow_{\mathsf{p}} \gamma_p \longrightarrow\!\!\!\!\!\rightarrow_{p} \gamma'_p$$
$$\implies \exists \gamma_q, \gamma'_q : \gamma \longrightarrow\!\!\!\!\!\rightarrow_{\mathsf{q}} \gamma_q \longrightarrow\!\!\!\!\!\rightarrow_{\mathsf{q}} \gamma'_q \ \wedge \ \mathsf{heap}(\gamma_p) \approx_{\Phi^{\mathsf{u}}} \mathsf{heap}(\gamma_q) \ \wedge \ \mathsf{heap}(\gamma'_p) \approx_{\Phi^{\mathsf{u}}} \mathsf{heap}(\gamma'_q)$$

3 Trace Equivalence

Definition 1 is difficult to establish, it requires considering a potentially infinite number of executions and deep inspection of the heap. In this section we propose a sufficient condition for Definition 1, which can be established by inspection of the stack only at certain points in each execution. We show this sufficient condition in Definition 2. We anticipate that this condition combined with suitable approximation techniques will allow us to make progress in applying static analysis to the problem of checking the correctness of refactorings.

In Definition 2 we will consider traces. A *trace* is the sequence of stack frames observable upon entry/exit of Φ^{u}. Traces for our running example are shown in Figure 3. In this case the elements of the trace correspond to the entry and exit points of the constructor, and the `add` and `remove` methods, of the class `FiFo`. Each trace element contains the address of the method receiver and the address of any parameters. On method return we use `ret` as a synthetic method name, and also to name the return value. For example, on entry to the `add` method we capture the address of `this` and the parameter `o`. On exit of the `remove` method we capture the address of the return value. The ith element of trace $\tau\mathsf{s}$ is written as $\tau\mathsf{s}_i$.

Figure 3 also shows examples of isomorphic and non-isomorphic traces. Two traces are equivalent if they have the same aliasing structure and `null` in the same places. Section 4 describes this in more detail. The judgement $\tau\mathsf{s} \approx \tau\mathsf{s}'$ holds whenever $\tau\mathsf{s}$ and $\tau\mathsf{s}'$ are isomorphic modulo addresses.

Definition 2 gives the formal meaning of trace equivalent executions. It holds whenever, for every initial state, execution in p produces an equivalent trace with execution in q. We write $\gamma \xrightarrow{\tau\mathsf{s}}\!\!\!\!\!\rightarrow_{p} \gamma'$ to mean the multi-step execution of program p from configuration γ to configuration γ' with trace $\tau\mathsf{s}$. Trace concatenation is performed by the $\cdot$ function.

▶ **Definition 2** (Trace Equivalent Executions). p and q are *trace equivalent* iff for every state reachable from an initial state with some trace in p (q), a state is reachable from the same initial state in q (p) with an equivalent trace.

p and q are trace equivalent when:

$$\forall\, \tau\mathsf{s}_p, \tau\mathsf{s}_q, \gamma, \gamma_p, \gamma_q : \mathsf{init}(\gamma) \ \wedge \ \gamma \xrightarrow{\tau\mathsf{s}_p}\!\!\!\!\!\rightarrow_{\mathsf{p}} \gamma_p \ \wedge \ \gamma \xrightarrow{\tau\mathsf{s}_q}\!\!\!\!\!\rightarrow_{\mathsf{q}} \gamma_q$$
$$\implies \left(\begin{array}{l} \left(\exists\, \tau\mathsf{s}'_p, \gamma'_p : \gamma_p \xrightarrow{\tau\mathsf{s}'_p}\!\!\!\!\!\rightarrow_{\mathsf{p}} \gamma'_p \wedge \tau\mathsf{s}_p \cdot \tau\mathsf{s}'_p \approx \tau\mathsf{s}_q \right) \ \vee \\ \left(\exists\, \tau\mathsf{s}'_q, \gamma'_q : \gamma_q \xrightarrow{\tau\mathsf{s}'_q}\!\!\!\!\!\rightarrow_{\mathsf{q}} \gamma'_q \wedge \tau\mathsf{s}_p \approx \tau\mathsf{s}_q \cdot \tau\mathsf{s}'_q \right) \end{array} \right)$$

Lemma 3 relates Definition 1 and Definition 2. It directly relates equivalence of traces to equivalence of heaps wrt. Φ^{u}. In particular it says that whenever executions of p and q produce equivalent traces, then they will also correspond in Φ^{u}. Section 6 sketches a proof of Lemma 3.

▶ **Lemma 3.** *If* p *is trace equivalent with* q *(Definition 2) then* Φ^{m} *bounds the modification (Definition 1).*

4 Partitions and Traces

Heap partition is defined by the function part : Object $\rightarrow\{\Phi^{\mathsf{m}}, \Phi^{\mathsf{u}}\}$, which gives a partition identifier for any object. Objects must not move between partitions during execution. The judgement mod(o) holds iff the object o is modified. An *object is modified* iff its class is modified, and a class is modified iff any of its methods' bodies or its fields differ between p and q. All modified objects must be in the Φ^{m} partition, but the Φ^{m} may also contain non-modified objects[1].

We say that Φ^{u} is entered whenever the this pointer of the top stack frame points into Φ^{m} and a stack frame is pushed or popped leaving a top stack frame whose this pointer points into the Φ^{u}. And conversely for exit of Φ^{u}. For example, in Figure 1 if the top stack frame is popped execution will enter Φ^{u}.

$\tau\mathsf{s}_p$

$\tau\mathsf{s}_{p1}$	Fifo	this:	3	
$\tau\mathsf{s}_{p2}$	ret	ret:	3	
$\tau\mathsf{s}_{p3}$	add	this:	3	o: 1
$\tau\mathsf{s}_{p4}$	ret			
$\tau\mathsf{s}_{p5}$	add	this:	3	o: 2
$\tau\mathsf{s}_{p6}$	ret			
$\tau\mathsf{s}_{p7}$	remove	this:	3	
$\tau\mathsf{s}_{p8}$	ret	ret:	1	
$\tau\mathsf{s}_{p9}$	remove	this:	3	
$\tau\mathsf{s}_{p10}$	ret	ret:	2	

$\tau\mathsf{s}_q$

$\tau\mathsf{s}_{q1}$	Fifo	this:	10	
$\tau\mathsf{s}_{q2}$	ret	ret:	10	
$\tau\mathsf{s}_{q3}$	add	this:	10	o: 8
$\tau\mathsf{s}_{q4}$	ret			
$\tau\mathsf{s}_{q5}$	add	this:	10	o: 9
$\tau\mathsf{s}_{q6}$	ret			
$\tau\mathsf{s}_{q7}$	remove	this:	10	
$\tau\mathsf{s}_{q8}$	ret	ret:	8	
$\tau\mathsf{s}_{q9}$	remove	this:	10	
$\tau\mathsf{s}_{q10}$	ret	ret:	9	

$\tau\mathsf{s}_r$

$\tau\mathsf{s}_{r1}$	Fifo	this:	7	
$\tau\mathsf{s}_{r2}$	ret	ret:	7	
$\tau\mathsf{s}_{r3}$	add	this:	7	o: 4
$\tau\mathsf{s}_{r4}$	ret			
$\tau\mathsf{s}_{r5}$	add	this:	7	o: 6
$\tau\mathsf{s}_{r6}$	ret			
$\tau\mathsf{s}_{r7}$	remove	this:	7	
$\tau\mathsf{s}_{r8}$	ret	ret:	6	
$\tau\mathsf{s}_{r9}$	remove	this:	7	
$\tau\mathsf{s}_{r10}$	ret	ret:	4	

$\tau\mathsf{s}_p \approx^{\beta} \tau\mathsf{s}_q$ when $\beta = (1,8),(3,10),(2,9)$ $\qquad \tau\mathsf{s}_q \not\approx \tau\mathsf{s}_r$

Figure 3 Three traces. The trace $\tau\mathsf{s}_p$ is from an execution of p. The trace $\tau\mathsf{s}_q$ is from an execution of q. The trace $\tau\mathsf{s}_r$ is fictional and does not come from either p or q. Trace $\tau\mathsf{s}_p$ is equivalent to trace $\tau\mathsf{s}_q$ under the address bijection, β, shown. Trace $\tau\mathsf{s}_r$ is not equivalent to either $\tau\mathsf{s}_p$ or $\tau\mathsf{s}_q$, there is no bijection between the addresses of $\tau\mathsf{s}_p$ and $\tau\mathsf{s}_r$ that also preserves the aliasing structure of the traces.

Each element of a trace contains the values of method parameters, or method return values, from the top stack frame whenever the partition is crossed. Only the address values, and associated parameter names, actually present in the stack frame are captured, no information from the heap is needed. Since the actual addresses used by any two executions can vary unpredictably we cannot compare traces directly. Instead we say that traces are equivalent if there exists a bijection β between the addresses present in each trace, that preserves the structure of the trace. Figure 3 shows an example of trace equivalence and non-equivalence.

We write $\tau\mathsf{s}_i(x)$ to mean the value at the location associated with the variable x in the ith element of trace $\tau\mathsf{s}$, or $\perp$ if the variable is not defined in that trace element, and $\tau\mathsf{s}_i(\mathsf{meth})$ to mean the name of the method associated with the ith element of trace $\tau\mathsf{s}$.

[1] For example, we may refactor some code to keep a list sorted for improved search performance. The code of list would not be modified, but its state in the heap would be affected.

The equivalence relation $\approx$ over traces, $\tau\mathsf{s}, \tau\mathsf{s}'$, holds if the traces contain the same number of elements, elements at the same position in the sequence are calls to methods with the same identifier, there is a structure preserving bijection between the addresses in each trace, and both traces have null at the same locations. Therefore, whenever $\tau\mathsf{s} \approx \tau\mathsf{s}'$ holds, whatever is an alias in $\tau\mathsf{s}$ is an alias in $\tau\mathsf{s}'$ and vice versa.

▶ **Definition 4.** The relation $\approx$ holds whenever two traces contain the same sequence of method names, and when the aliasing structure of both traces is precisely the same.

$$\approx^{\beta} \subseteq \mathsf{Trace} \times \mathsf{Trace}$$

$$\tau\mathsf{s} \approx^{\beta} \tau\mathsf{s}' \overset{\text{def}}{\iff} \forall i \in \mathbb{N}, x \in \mathrm{Id}_v : (\tau\mathsf{s}_i(\mathsf{meth}) = \tau\mathsf{s}'_i(\mathsf{meth}) \ \wedge \ \beta(\tau\mathsf{s}_i(x)) = \tau\mathsf{s}'_i(x)) \ \wedge$$
$$\beta(\mathsf{null}) = \mathsf{null} \ \wedge \ (\forall a, a' : \beta(a) = \beta(a') \implies a = a')$$

$$\tau\mathsf{s} \approx \tau\mathsf{s}' \overset{\text{def}}{\iff} \exists \beta : \tau\mathsf{s} \approx^{\beta} \tau\mathsf{s}'$$

5 Properties of Bounded modifications

In this section we describe some properties of executions when part has been picked such that Definition 1 holds, i.e. Φ^{m} bounds the modification of interest. The usefulness of definition 1 is that it helps a programmer to reason about heaps, stacks and executions. We now describe some properties of heaps, stacks and executions that follow from Lemma 3 and hold for bounded modifications.

The structure of executions wrt. the partitioning is a sequence of execution steps in one partition, followed by a sequence of execution steps in the other partition, then back to the original partition, and so on in an interleaved fashion as shown in Figure 4.

Corollary 5 describes the sequences of states that the Φ^{u} partition reaches when execution is in Φ^{m}. We write $\mathsf{part}(\gamma)$ to mean the partition of the object pointed to by the this pointer of the topmost stack frame of state γ. Figure 4 illustrates the preservation of Φ^{u} heap equivalence, when execution is in Φ^{m}, that the corollary describes. In particular, between corresponding parts of the executions in Φ^{m} all of the heaps are equivalent wrt. the Φ^{u} partition.

▶ **Corollary 5.** *When Φ^{m} bounds the modification, every state with execution inside Φ^{m} reachable with some trace in p, and every state reachable with some equivalent trace in q have equivalent heaps wrt. Φ^{u}.*

$$\forall \gamma, \gamma_p, \gamma_q, \tau\mathsf{s}_p, \tau\mathsf{s}_q : \mathsf{init}(\gamma) \ \wedge \ \gamma \xrightarrow{\tau\mathsf{s}_p}\!\!\!\!\!\rightarrow_{\mathsf{p}} \gamma_p \ \wedge \ \gamma \xrightarrow{\tau\mathsf{s}_q}\!\!\!\!\!\rightarrow_{\mathsf{q}} \gamma_q \ \wedge \ \tau\mathsf{s}_p \approx \tau\mathsf{s}_q \ \wedge \ \mathsf{part}(\gamma_p) = \Phi^{\mathsf{m}}$$
$$\implies \mathsf{heap}(\gamma_p) \approx_{\Phi^{\mathsf{u}}} \mathsf{heap}(\gamma_q)$$

Corollary 6 describes the sequences of states that the Φ^{u} partition reaches for execution in Φ^{u}. We write $\xrightarrow{\tau}_p$ to mean a single execution step in program p that produces the non-empty trace τ, and we write $\xrightarrow{\epsilon}_p$ to mean a single execution step in program p that produces an empty trace. We use the equivalence $\gamma \approx_{\Phi^{\mathsf{u}}} \gamma'$ to mean that the part of the heap consisting only of objects in Φ^{u} and the topmost stack frames of γ and γ' are isomorphic modulo addresses. Figure 4 illustrates the lockstep equivalence wrt. Φ^{u} when execution is within Φ^{u}.

In particular, between corresponding parts of the executions in Φ^{u}, the same number of steps are executed in p as q, and corresponding pairs of states are equivalent wrt. Φ^{u}. Intuitively, the two executions proceed in lockstep for as long as execution remains in Φ^{u}.

▶ **Corollary 6.** *When Φ^{m} bounds the modification, for every state reachable n steps after entering Φ^{u} with some trace in p, there is an equivalent state wrt. Φ^{u} reachable n steps after*

each entry to Φ^{u} *that is reachable with an equivalent trace in* q*, provided that none of the* n *steps in* p *crosses out of* Φ^{u}*.*

$$\begin{array}{l}\forall \gamma, n, \gamma_p, \gamma_{p1} \ldots \gamma_{pn}, \gamma_q, \gamma'_q, \tau\mathsf{s}_p, \tau_p, \tau\mathsf{s}_q, \tau'_q : \exists \gamma_{q1} \ldots \gamma_{qn} : \\ \quad \mathsf{init}(\gamma) \ \wedge \ \mathsf{part}(\gamma_{p1}) = \Phi^{\mathsf{u}} \ \wedge \ \tau\mathsf{s}_p \cdot \tau_p \approx \tau\mathsf{s}_q \cdot \tau_q \\ \quad \gamma \xrightarrow{\tau\mathsf{s}_p}\!\!\!\!\rightarrow_{\mathsf{p}} \gamma_p \xrightarrow{\tau_p}_{\mathsf{p}} \gamma_{p1} \xrightarrow{\epsilon}_{\mathsf{p}} \ldots \xrightarrow{\epsilon}_{\mathsf{p}} \gamma_{pn} \ \wedge \ \gamma \xrightarrow{\tau\mathsf{s}_q}\!\!\!\!\rightarrow_{\mathsf{q}} \gamma_q \xrightarrow{\tau_q}_{\mathsf{q}} \gamma'_q \\ \Longrightarrow \\ \quad \gamma'_q = \gamma_{q1} \ \wedge \ \gamma_{q1} \xrightarrow{\epsilon}_{\mathsf{q}} \ldots \xrightarrow{\epsilon}_{\mathsf{q}} \gamma_{qn} \ \wedge \ \bigwedge_{i=1}^{n} \gamma_{pi} \approx_{\Phi^{\mathsf{u}}} \gamma_{qi}\end{array}$$

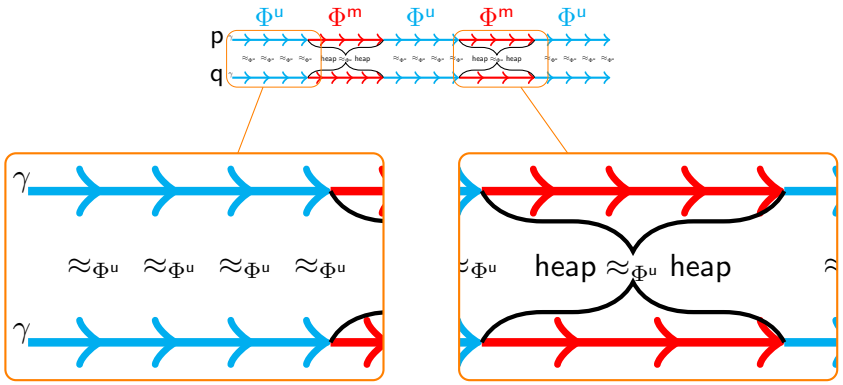


Figure 4 Two executions from initial state γ are shown. The top execution is of program p and the bottom execution is of q. The modification between p and q is assumed to be bounded by Φ^{m}. The leftmost magnified area shows the step wise correspondence of execution steps when execution is within Φ^{u}. The pairs of states are stepwise equivalent wrt. Φ^{u}, in accordance with Corollary 6. The rightmost magnified area shows the preservation of heap equivalence wrt. Φ^{u} whilst execution is in Φ^{m}, in accordance with Corollary 5.

6 Proof Sketch

We provide two auxiliary lemmas that support Lemma 3. Lemma 7 says that for as long as an execution remains outside of a partition, it will not affect the state of any heap objects inside of the partition. We write $\mathsf{heap}(\gamma)_\phi$ to mean the the heap of state γ with objects outside ϕ deleted (leaving dangling pointers if necessary). We justify this lemma by noting that: privacy of fields prevents manipulation of the state of a partition unless the this pointer is pointing inside that partition; entering the partition would cause the trace to be non-empty.

▶ **Lemma 7.** *As long as the execution is outside the partition the heap inside the partition is preserved.*

$$\forall p, \gamma, \gamma', \phi : \gamma \xrightarrow{\epsilon}\!\!\!\!\rightarrow_p \gamma' \ \wedge \ \mathsf{part}(\gamma) \neq \phi \implies \mathsf{heap}(\gamma)_\phi = \mathsf{heap}(\gamma')_\phi$$

Lemma 8 says that the behaviour of an execution while it is in a partition ϕ is not affected by the state of the heap outside of that partition. We write γ_ϕ to mean the state which is the same as γ, but with any objects not in ϕ deleted from the heap. In particular that, unless execution leaves ϕ, the heap inside the partition will reach the same heap states wrt. the partition dependent only on the state inside the partition. We justify this lemma by noting that: field privacy prevents observation of the state of a partition unless execution is

in the partition; execution is deterministic; and the continuations of γ and γ_ϕ are the same. Therefore, only the aliasing structure of pointers inside the partition (including those that point out of the partition), and the code of the classes of objects inside the partition, affects the execution inside the partition.

▶ **Lemma 8.** *The state of the heap outside a partition does not affect execution inside the partition. In particular, if all the objects outside of the partition ϕ were deleted from the heap, execution will still reach equal states wrt. ϕ as long as execution does not leave ϕ.*

$$\forall p, \phi, \gamma, \gamma' \exists \gamma'' : \mathsf{part}(\gamma) = \phi \ \wedge \ \gamma \xrightarrow{\epsilon} \!\!\!\!\rightarrow_p \gamma' \implies \gamma_\phi \xrightarrow{\epsilon} \!\!\!\!\rightarrow_p \gamma'' \ \wedge \ \gamma'_\phi = \gamma''$$

7 Conclusion

We have argued that information about aliasing in partition crossing calls is sufficient to check the propagation of heap effects between areas of the heap. We have presented a definition of what it means for the heap effect of a program modification to be contained within a programmer defined heap partition.

We intend to continue this work by performing a proof of the lemmas given in this paper. We will then try to extend this work to characterise program modifications that are not behaviour preserving.

References

1 Fabian Bannwart and Peter Müller. Changing programs correctly: Refactoring with specifications. *FM 2006: Formal Methods*, pages 492–507, 2006.

2 Ran Ettinger. Program sliding. In James Noble, editor, *ECOOP 2012 Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science*, pages 713–737. Springer Berlin Heidelberg, 2012.

3 B. Godlin and O. Strichman. Regression verification. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 466 –471, july 2009.

4 Chris Hawblitzel, Ming Kawaguchi, Shuvendu K Lahiri, and Henrique Rebêlo. Towards modularly comparing programs using automated theorem provers. *International Conference on Automated Deduction*, June 2013.

5 Kevin J. Hoffman, Patrick Eugster, and Suresh Jagannathan. Semantics-aware trace analysis. In *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '09, pages 453–464, New York, NY, USA, 2009. ACM.

6 Shuvendu Lahiri, Ken McMillan, Rahul Sharma, and Chris Hawblitzel. Differential assertion checking. In *Foundations of Software Engineering*. ACM, 2013.

7 Leonardo Mariani, Fabrizio Pastore, and Mauro Pezze. Dynamic analysis for diagnosing integration faults. *IEEE Trans. Softw. Eng.*, 37(4):486–508, July 2011.

8 Ivan Moore. Automatic inheritance hierarchy restructuring and method refactoring. In *ACM SIGPLAN Notices*, volume 31, pages 235–250. ACM, 1996.

9 David Naumann and Anindya Banerjee. State based encapsulation for modular reasoning about behaviour-preserving refactorings. In Dave Clarke, James Noble, and Tobias Wrigstad, editors, *Aliasing in Object-oriented Programming*. Springer State-of-the-art Surveys, 2012.

10 William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, 1992.

11 Javier Perez, Yania Crespo, Berthold Hoffmann, and Tom Mens. A case study to evaluate the suitability of graph transformation tools for program refactoring. *Software Tools for Technology Transfer - Special Section on GraBaTs 08*, 12(3-4):183–199, 2009. (c) Springer, 2009.

12 Max Schäfer, Mathieu Verbaere, Torbjörn Ekman, and Oege Moor. Stepping stones over the refactoring rubicon. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, pages 369–393, Berlin, Heidelberg, 2009. Springer-Verlag.

13 Yannick Welsch and Arnd Poetzsch-Heffter. Verifying backwards compatibility of object-oriented libraries using boogie. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, FTfJP '12, pages 35–41, New York, NY, USA, 2012. ACM.

Using Self-learning and Automatic Tuning to Improve the Performance of Sexual Genetic Algorithms for Constraint Satisfaction Problems*

Hu Xu[1], Karen Petrie[2], and Iain Murray[3]

1 **Computing School,**
QMB 1.10, University of Dundee
huxu@computing.dundee.ac.uk
2 **Computing School,**
QMB 2.10, University of Dundee
karenpetrie@computing.dundee.ac.uk
3 **Computing School,**
QMB 2.21, University of Dundee
irmurray@computing.dundee.ac.uk

Abstract

Currently the parameters in a constraint solver are often selected by hand by experts in the field; these parameters might include the level of preprocessing to be used and the variable ordering heuristic. The efficient and automatic choice of a preprocessing level for a constraint solver is a step towards making constraint programming a more widely accessible technology. Self-learning sexual genetic algorithms are a new approach combining a self-learning mechanism with sexual genetic algorithms in order to suggest or predict a suitable solver configuration for large scale problems by learning from the same class of small scale problems. In this paper, Self-learning Sexual genetic algorithms are applied to create an automatic solver configuration mechanism for solving various constraint problems. The starting population of self-learning sexual genetic algorithms will be trained through experience on small instances. The experiments in this paper are a proof-of-concept for the idea of combining sexual genetic algorithms with a self-learning strategy to aid in parameter selection for constraint programming.

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases Self-learning Genetic Algorithm, Sexual Genetic algorithm, Constraint Programming, Parameter Tuning

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.128

1 Introduction

The selection of suitable preprocessing levels for a given constraint problem is an important part of constraint programming (CP); efficiently tuning a constraint solver will shorten the search time and reduce the running cost. One significant method of increasing the search speed for a constraint solver is by tuning the solver's parameters [8]. Currently, the job of tuning the parameters is done manually; a skilled user selects the most suitable preprocessing method using previous experience from similar classes of problems. In most cases, the best preprocessing method used in similar classes of problems will provide a useful clue to aid the

* This work was funded by the School of Computing, University of Dundee.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andew V. Jones, Nicholas Ng; pp. 128–135

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

user's selection. However, this learning curve could be a barrier to novice users in learning how to efficiently use a CP solver [9].

Genetic algorithms (GA) are a classic global optimisation method posed by John Holland [6], which mimic the competition of organisms in nature and the mechanisms of evolution. GAs are usually implemented in a computer simulation in which a population of abstract representations of candidate solutions to an optimisation problem evolves towards better solutions. GAs are widely applied to optimisation problems such as function optimisation. Ansótegui et al. have proposed a gender-based genetic algorithm for the automatic configuration of algorithms[1] ; it shows that the sexual genetic algorithm (SGA) is feasible for automatic configuration.

In this paper, sexual genetic-based algorithms were chosen to select a preprocessing method for constraint satisfaction problems. There are three main reasons to choose SGAs to optimise preprocessing selection:

- SGAs have a powerful ability to tackle optimisation problems which lack auxiliary information
- SGAs perform parallel search rather than linear search; each chromosome (solution to the problem) competes against others in each generation
- SGAs are more efficient than standard GAs in preprocessing selection; it is not necessary for the SGA to evaluate the fitness of all chromosomes, which is a considerable consumer of CPU time.

Therefore the idea of combining SGAs and constraint programming seems worth exploring further and it is expected that automatic tuning will lead to improvements over manual tuning by users. ParamILS and CALIBRA [7] have demonstrated the practicality and efficiency of automatic configuration for constraints solvers. However, the general framework of combining GAs with constraint programming and the exploration of parameter sensitivity of GAs to any problems, has not been achieved. In light of this situation, a self-learning sexual genetic-based method for tuning Minion [4], which is one of the most efficient constraint solvers in the world, was proposed.

This paper firstly investigates an SGA and explores its features. The efficiency of this SGA will be tested by comparison with a standard GA for the Travelling Salesman Problem. The self-learning genetic algorithm (SLGA) will then be introduced and applied to select the preprocessing level. Finally, the efficiency of the self-learning sexual genetic algorithm (SLSGA) will be analysed. This paper also provides a proof for one possibility of improving sexual genetic algorithms for preprocessing selection in constraint satisfaction problems [2].

2 Sexual Genetic Algorithm

The basic concepts and features of SGAs are similar to standard GAs. As in standard GAs, there are three basic operators in sexual genetic algorithms: selection, crossover and mutation. The selection, which decides the parents for mating, is very important for evolution. In standard genetic algorithms, there is just one selection strategy during evolution. In nature, male individuals try to spread their gene information as widely as possible and female individuals try to select the fittest males to mate with [13]. Inspired by the natural behaviour of male vigor and female choice, sexual genetic algorithms [10] [12] apply two different selection mechanisms: male group (competitive) and female group (co-operative).

The first step of a genetic algorithm (GA) is called the encoding which is to construct a suitable starting chromosome for the optimisation problem and which can transfer solutions of the optimisation problem to the child chromosomes, where each child chromosome presents

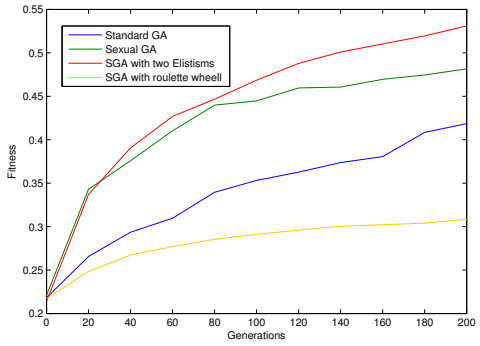


Figure 1 The efficiency of sexual genetic algorithm in solving the Travelling Salesman Problem.

one possible solution. Fitness describes the ability of an individual to reproduce in biology; the fitness function evaluates the difference between the desired result and the actual result.

Selection in genetic algorithms is the strategy which allows the best parents (with highest fitness) to have an increased chance of being selected to generate the next generation. Roulette wheel selection is a commonly-used way of choosing individuals from the population of chromosomes in a way that is proportional to their fitness. Roulette does not guarantee that the fittest member goes through to the next generation, merely that it has a better chance of doing so. In sexual genetic algorithms, the population is randomly divided into to two groups: male (competitive) and female (co-operative) as in nature. The male chromosomes have to compete for the chance of mating (the elitisms (the chromosomes with better fitness) will confer an increased chance of mating), while the female chromosomes have the same opportunity for mating. The running time of the fitness evaluation is substantial when automated tuning is used, because the fitness (searching time) of each chromosome has to be calculated with a given set of preprocessing for the constraint problem. The pseudocode (Algorithm 1) shows that half of the population is selected as male, meaning that half of the fitness evaluation time is saved while the variety of the population is maintained. This is the most important reason that sexual genetic algorithms were selected for the experiment in this paper.

Crossover can improve the fitness of the whole population quickly by mating parents to produce an offspring. Single-point crossover is the most common crossover in genetic algorithms because it can be easily understood and realized. The single crossover will be applied in sexual genetic algorithm.

Mutation, which changes one or more genes in an individual, is another operator used in GA. Mutation can help genetic algorithms escape the local maximum state by creating a new gene string. All of the mutations in this paper's experiments are single-point mutations.

3 Sexual Genetic Algorithm vs. Standard Genetic Algorithm by Solving TSP

The Travelling Salesman Problem (TSP) seeks the shortest Hamiltonian cycle path between n given cities and is a classic NP-complete problem. To prove the efficiency of sexual genetic algorithms and explore their features, a sexual genetic algorithm was applied to solve the TSP with different elitism percentages and this was compared with a standard genetic algorithm. There are three elitism strategies: one elitism, two elitisms and roulette wheel selection elitism.

Table 1 The solving times of different constraint satisfaction problems using of a sexual genetic algorithm.

	BIBD (Solving Time)	Langford (Solving Time)
Sexual GA	1.7 s	3.2 s
Standard GA	3.5 s	4.3 s

Figure 1 shows the efficiency of an SGA to solve the TSP. There are four curves in the graph: standard GA, SGA with one elitism, two elitisms and roulette wheel selection elitisms in male competition. It clearly shows that the SGA with two elitisms in male competition is most the efficient in solving the TSP; however, more elitisms does not mean better evolutionary speed because more elitisms could lead to high convergence. Compared with the standard GA, the SGA with optimised elitisms in male competition always approaches a better fitness.

Finally, the SGA was used to choose the best preprocessing method for constraint problems. In this paper, two classic constraint optimisation problems, balanced incomplete block design (BIBD) and the Langford's Number problem[1] , were chosen as the optimisation problem for testing the SGA. Following David's MicroGA Settings[3], the crossover rate is 0.5 and the mutation rate is 0.04 in all experiments. Each trial was run 100 times and the average of the minima was recorded.

Table 1 shows the solving times of the SGA for different problems in comparison with a standard GA. It shows that the SGA could find better preprocessing methods for both the BIBD and Langford's Number problems than standard GA for a small number of generations.

4 Self-learning Sexual Genetic Algorithm vs. Self-learning Genetic Algorithm

Generally, machine learning makes predictions by training, validation and testing itself against existing data [11] . Self-learning, which learns its own inductive bias based on previous experience, is one of the typical algorithms in the machine learning domain. Self-learning could avoid repetition of searching and computation in the previous experiments. In genetic algorithms, the generation of the starting population is a considerable factor, as are the crossover rate and mutation rate.

To check the influence of the fitness value in the starting population of genetic algorithms, two different starting populations (set to high fitness and low fitness) were applied to optimise the same function $F(x) = x^{10}$[5] where $0 < x < 1$. The size of both starting populations was 30 chromosomes. Figure 2 shows that the genetic algorithm with a starting population with high fitness could approach better fitness than the one with a starting population with poor fitness. This demonstrates that a suitable starting population for a genetic algorithm could lead to a more rapid approach to best fitness, and suggests that this could be applied to self-learning in machine learning.

A self-learning genetic algorithm (SLGA) [14] is an algorithm which makes the preprocessing prediction by using previous experience on the same classes of constraint satisfaction problem. An SLGA can improve the search speed by defining a specific starting population

[1] All From www.csplib.org

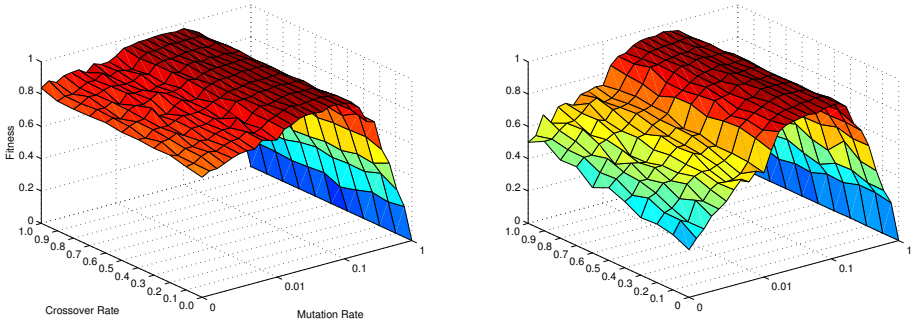


Figure 2 The evolutionary speed comparison with different starting populations. The X axis is the mutation rate and the Y axis the Crossover rate. The Z axis is the best fitness after 50 generations with different mutation rate and crossover rate. The fitness of the starting population of the left graph is lower than 0.1. The fitness of the starting population in the right graph is randomly generated between 0 and 1.

(instead of a random starting population as normally used), with the starting population taken from the training data of the same class of small instance problems.

4.1 Self-learning Sexual Genetic Algorithm for Constraint Satisfaction Problem

The SLSGA pseudocode (Algorithm 1) introduces SLSGA's working principle and clearly shows how self-learning combines with the sexual genetic algorithm. Compared with the self-learning standard genetic algorithm, the self-learning sexual genetic algorithm halves the time spent on fitness evaluation and selects k elitisms from the male group, instead of from the whole population. Compared with the SGA, the SLSGA also has an optimised starting population; the SLSGA is trained using small scale problems to select the starting population for large scale problems.

Algorithm 1 Self-learning Sexual Genetic Algorithm

Produce the starting populations P_i for small instance problem from the experience of small instance ▷ i is the population size
for $j = 1$ to n **do** ▷ j is the generation
  repeat
    Randomly select m chromosomes of population as male
    The rest chromosomes is selected as female ▷ $m = i/2$ in our SGA for tuning constraint programming solver
    Evaluate the fitness of male chromosomes
    Select k elitisms from male chromosomes to mating pool
    Each female chromosomes has the same possibility for mating
    New generation is created from k elitisms and mother chromosomes by crossover and mutation
  until λ =The best preprocessing found or the searching time is out of time limit
end for
return λ

Table 2 The solving times of Self-Learning Sexual Genetic Algorithm. "CSP Problems" refers to the training instance and the optimised problems. "Training Instance" is the search time in seconds and total search nodes for small instances without preprocessing. "Target Instance" is the search time and total search nodes for optimised (large) instances without preprocessing. "Sexual GA" and "SLSGA" are the search time and total search nodes with preprocessing that had been optimised with Sexual Genetic Algorithm and Self-learning Sexual Genetic Algorithm.

CSP Problems	Training Instance		Target Instance		Sexual GA		SLSGA	
	Running Time	Nodes	Running Time	Nodes	Running Time	Nodes	Running Time	Nodes
BIBDline1 BIBDline6	0.75	4332	3.67	48502	3.671	48502	2.5	32773
BIBDline1 BIBDline8	0.75	4332	6.56	90432	5.21	11	3.8	11
BIBDline2 BIBDline8	1.21	10952	6.56	90432	3.81	25269	3.7	25269
OpenStack1 OpenStack2	0.156	106	13.2	835567	9.9	15828	9.9	15828
Lanford(3,10) Lanford(2,20)	0.16	60	22	280968	1.95	22	0.21	696
Lanford(3,10) Lanford(3,17)	0.16	60	13.2	105873	2.13	11840	1.59	8729
BIBDline4 BIBDline11	2.3	29257	207	2287940	136	1670535	136	1670535

The experimental results of SLSGA show that an optimised starting population easily and quickly leads to a better evolutionary result. The aim of automatic tuning is to use the shortest time possible to find the optimal preprocessing settings. To prove the correctness of the hybridization idea (combining self-learning with SGA) and the efficiency of SLSGA, it was then applied to solve some constraint satisfaction problems with different instances: BIBD, Langford's Number and Open Stack problems. All the settings used were the same as others described in this paper.

Table 2 shows the efficiency of SLSGA for solving various constraint satisfaction problems. The table shows the number of search nodes and running time for the solver to find the solution after optimisation of SGA and SLSGA. It clearly shows that SLSGA could arrive at acceptable result more efficiently than SGA, improving the evolutionary speed and deriving useful results at the same time. This shows that self-learning is a feasible algorithm for preprocessing selection in constraint satisfaction problems, although the optimal result from SLSGA is not significantly better than that from SGA.

5 Conclusions and Future work

To prove the concept and potential effectiveness of self-learning sexual genetic algorithms, this paper has firstly shown the efficiency of a sexual genetic algorithm by solving the TSP and comparing this with a standard genetic algorithm. It has demonstrated that the elitisms percentage in the SGA is very important and that selecting suitable elitisms percentages leads to an ideal optimisation speed. Self-learning was proposed to improve the tuning efficiency from previous experience (in the same way as human behaviour) rather than by logical inference. The experimental results showed that the large scale problem could be properly solved using an optimised starting population which was trained using data from small scale problems. Thus the self-learning sexual genetic algorithm was able to achieve satisfactory results by combining two strategies.

The results show that self-learning genetic algorithms can be efficient methods for selecting preprocessing of constraint-solving problems. However, a number of challenges remain for future exploration. In this paper, four classic problems were used to verify the efficiency of a self-learning sexual genetic algorithm on large problems. More and larger-scale problems such as the car sequence problem will be used to explore the efficiency and limitations of SLSGAs. With regard to self-learning, the convergence of the starting population is worth further exploration, for example whether having many optimised individuals in the starting population can lead to local traps. The application of the self-learning strategy to deal with multiple instances is also of interest.

Currently the best model to solve a constraint satisfaction problem is selected by hand by the user; this paper has shown that improved performance may be obtained by applying self-learning sexual genetic algorithms to constraint satisfaction problems.

References

1 C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September20-24, 2009 Proceedings*, page 142. Springer, 2009.

2 David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study.* Princeton University Press, 2011.

3 David L. Carroll. Chemical laser modeling with genetic algorithms. *Aiaa Journal*, 34:338–346, 1996.

4 Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In *Proceedings of the 17th Eureopean Conference on Artificial Intelligence (ECAI'06)*, pages 98–102, 2006.

5 David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

6 John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. 1992.

7 Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1152–1157. AAAI Press, 2007.

8 Lars Kotthoff, Ian Miguel, and Peter Nightingale. Ensemble classification for constraint solver configuration. In *Principles and Practice of Constraint Programming–CP 2010*, pages 321–329. Springer, 2010.

9 Tony Lambert, Carlos Castro, Eric Monfroy, María Riff, and Frédéric Saubion. *Hybridization of Genetic Algorithms and Constraint Propagation for the BACP*, volume 3668 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005.

10 M.M. Raghuwanshi and O.G. Kakde. Genetic algorithm with species and sexual selection. In *Cybernetics and Intelligent Systems, 2006 IEEE Conference on*, pages 1–8, 2006.

11 Simon Rogers and Mark Girolami. *A First Course in Machine Learning*. Chapman & Hall/CRC, 1st edition, 2011.

12 M. Jalali Varnamkhasti and MasoumehVali. Sexual selection and evolution of male and female choice in genetic algorithm. *Scientific Research and Essays*, 7(31):2788–2804, 2012.

13 Stefan Wagner and Michael Affenzeller. Sexualga: Gender-specific selection for genetic algorithms. In *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI)*, volume 4, pages 76–81. Citeseer, 2005.

14 Hu Xu and Karen Petrie. Self-Learning Genetic Algorithm For Constrains Satisfaction Problems. In Andrew V. Jones, editor, *2012 Imperial College Computing Student Workshop*, volume 28 of *OpenAccess Series in Informatics (OASIcs)*, pages 156–162, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Achieving Superscalar Performance without Superscalar Overheads – A Dataflow Compiler IR for Custom Computing

Ali Mustafa Zaidi and David J. Greaves

University of Cambridge Computer Laboratory
Cambridge CB3 0FD, UK
{ali-mustafa.zaidi, david.greaves}@cl.cam.ac.uk

Abstract

The difficulty of effectively parallelizing code for multicore processors, combined with the end of threshold voltage scaling has resulted in the problem of 'Dark Silicon', severely limiting performance scaling despite Moore's Law. To address dark silicon, not only must we drastically improve the energy efficiency of computation, but due to Amdahl's Law, we must do so without compromising sequential performance. Designers increasingly utilize custom hardware to dramatically improve both efficiency and performance in increasingly heterogeneous architectures. Unfortunately, while it efficiently accelerates numeric, data-parallel applications, custom hardware often exhibits poor performance on sequential code, so complex, power-hungry superscalar processors must still be utilized. This paper addresses the problem of improving sequential performance in custom hardware by (a) switching from a statically scheduled to a dynamically scheduled (dataflow) execution model, and (b) developing a new compiler IR for high-level synthesis that enables aggressive exposition of ILP even in the presence of complex control flow. This new IR is directly implemented as a static dataflow graph in hardware by our high-level synthesis tool-chain, and shows an average speedup of 1.13× over equivalent hardware generated using LegUp, an existing HLS tool. In addition, our new IR allows us to further trade area & energy for performance, increasing the average speedup to 1.55×, through loop unrolling, with a peak speedup of 4.05×. Our custom hardware is able to approach the sequential cycle-counts of an Intel Nehalem Core i7 superscalar processor, while consuming on average only 0.25× the energy of an in-order Altera Nios IIf processor.

1998 ACM Subject Classification B.5.1 Design, B.6.1 Design Styles, B.6.3 Design Aids, C.1.3 Other Architecture Styles, D.3.2 Language Classifications, D.3.4 Processors

Keywords and phrases High-level Synthesis, Instruction Level Parallelism, Custom Computing, Compilers, Dark Silicon

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.136

1 Introduction

Despite ongoing exponential growth of on-chip resources with Moore's Law, the performance scalability of future designs will be increasingly restricted. This is because the total *usable* on-chip resources will be growing at a much slower rate, due to the recently identified problem of Dark Silicon [6, 7]. Esmaelzadeh et al. identify two main sources of Dark Silicon:

1. *Amdahl's Law*: With the exception of certain numeric, data-parallel applications, most applications in the general-purpose domain lack sufficient *explicit* parallelism to make full use of the ever increasing number of cores on chip [1]. Due to Amdahl's Law, the overall speed-up for such applications is strictly constrained by sequential performance.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andew V. Jones, Nicholas Ng; pp. 136–144

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2. *Utilization Wall*: Even when an embarrassingly parallel application is not limited by Amdahl's Law, overall performance scaling will still be limited by the Utilization Wall [7, 17]: with each process generation, a decreasing fraction of on-chip transistor resources may be switched at full speed at one time, in order to meet the power budget.

Recent work has exploited custom (and reconfigurable) hardware to improve the per-core energy efficiency for the general-purpose application domain [17, 3]. Unfortunately, sequential code often exhibits much lower performance in custom hardware than a typical out-of-order superscalar processor [2, 3, 17]. For the general-purpose domain, sequential code involves irregular memory accesses and complex control-flow (e.g. nested loops, data-dependent branching). Currently the only means of achieving high performance on such code is through the use of complex, inefficient out-of-order superscalar processors. Such processors exhibit an exponential increase in power dissipation as performance increases [8].

This puts us between a rock and a hard place: without utilizing complex processors, performance scaling is limited by Amdahl's Law and poor sequential performance, but with such processors, the Utilization Wall limits speed-up by limiting the total active resources at any time. Esmaelzadeh et al. [6] focused on desktop, server and workstation domains, that have a reasonable power budget of 20-200W. This problem is exacerbated even further when we consider the increasingly important and rapidly growing portable computing domain, where power budgets are further limited to only 0.5-5W.

Our goal is to enable much more pervasive utilization of custom or reconfigurable hardware for general-purpose computation in order to mitigate the effects of dark silicon. For this, we must (a) overcome the performance limitations on sequential code in custom hardware, (b) without compromising its inherent energy-efficiency, while (c) requiring minimal programmer effort (i.e. minimal or no alterations to the source code or programming language).

To this end, we develop a new compiler intermediate representation (IR), called the Value State Flow Graph (VSFG), that exposes instruction-level parallelism (ILP) from sequential code even in the presence of complex control-flow. The VSFG is also designed to be directly implementable as custom hardware, replacing the traditionally used Control-Data Flow Graph (CDFG) [16]. To test our new IR, we have implemented a new high-level synthesis (HLS) tool-chain, that compiles from LLVM [13] to the VSFG, then implements it as a Verilog hardware description. Unlike the statically-scheduled execution model of traditional custom hardware, we employ the dynamically-scheduled 'Spatial Computation' model [3], in order to match the dynamic scheduling advantages of out-of-order processors, allowing for better tolerance of variable latencies and statically unpredictable behaviour.

2 Enhancing sequential performance in Custom Hardware

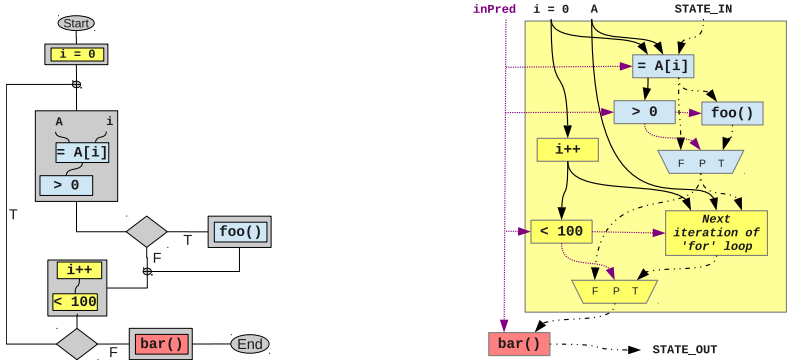
There are two main reasons that out-of-order superscalar processors are able to achieve higher performance on control-flow intensive sequential code [15]:

- Aggressive control-flow speculation to exploit ILP from across multiple basic-blocks, and
- Dynamic execution scheduling of instructions, approximating the dynamic dataflow execution model at runtime.

The Superscalar performance advantage: A Case Study. Listing 1 presents a code sample, and Figure 1a presents its equivalent CDFG (blue blocks represent '*if*' statement operations, while yellow blocks form the remainder of the *for* loop). Branch prediction in a superscalar processor will be able to overcome the control-flow dependences between the three basic blocks

Listing 1 Example C Code

```
for (i = 0; i < 100; i++)
  if (A[i] > 0) foo();
bar();
```



(a) The CDFG for Listing 1.

(b) The VSFG for Listing 1.

Figure 1 The equivalent CDFG and VSFG for the code given in Listing 1.

composing the for-loop. Furthermore, branch prediction enables *dynamic* unrollling of the for-loop, and exploitation ILP from across multiple iterations. Conventional processors make use of a centralized in-order commit buffer (i.e. *re-order buffer*) to selectively commit executed instructions in the correct CDFG order. In case of misprediction, executed instructions from mispredicted paths can simply be discarded, preserving correct program state.

On the other hand, custom hardware lacks the mechanisms to perform such aggressive control-flow speculation: unlike a conventional processor, there is no centralized commit buffer that can be used for misspeculation recovery. In the case of some forward (i.e. *if-else*) branches, it is sometimes possible to employ *if-conversion* to perform speculation by executing both sides of the branch and then discarding the result from the false path. However, this technique is limited to the simple cases where only data-flow is involved – the *if* branch in Listing 1 may invoke a separate function with its own control and data flow, and hence cannot simply be if-converted for speculation in hardware.

As it is not possible to perform speculation on backwards branches (i.e. loops) in custom hardware, High-level synthesis tools attempt to overcome this limitation by statically unrolling (as shown in Figure 2), flattening and pipelining loops in order to decrease the number of backwards branches that would be dynamically executed, but this can significantly increase the complexity of the centralized finite-state machines, resulting in very long combinational paths that can overwhelm any gains in ILP [11, 9].

In addition to aggressive control-flow speculation, superscalar processors employ dynamic execution scheduling, which helps in dealing with unpredictable behavior at runtime. Instructions are allowed to execute as soon as their input operands, as well as the appropriate execution resources, become available. Processors can even have multiple instances of the same instruction (say from a tightly wound loop) in flight, with their results written to different locations via register renaming. Using renaming, contemporary processors are able to approximate the dynamic-dataflow model of execution, allowing them to easily adapt to runtime variability to improve performance. For instance, even in the case that the load ("=

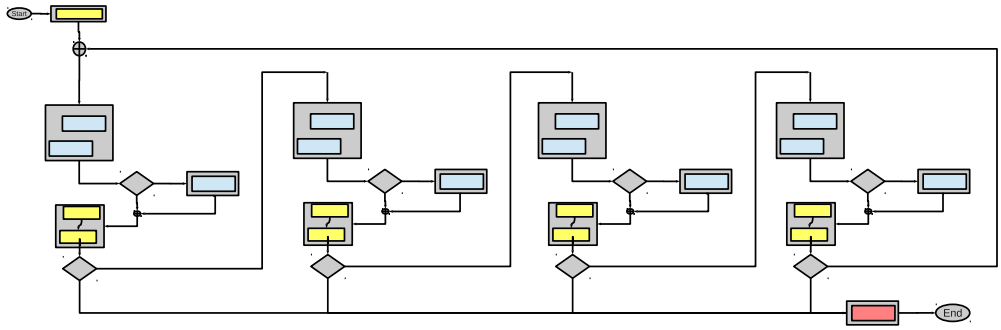


Figure 2 The CDFG from Figure 1a with the loop unrolled 4 times.

A[i]") instruction encounters a cache miss, all the remaining operations that are in flight from the multiple blocks and loop iterations may still execute in dataflow order - only those operations that are dependent on the value of the load will be delayed.

On the other hand, custom hardware employs static execution scheduling, where the execution schedule for operations is determined at compile-time, and implemented at run-time by a centralized state machine. This means that such hardware can only be conservatively scheduled for the multiple possible control-flow paths through the code, leaving it unable to adapt to runtime variability that may occur due to data-dependent control-flow, variable-latency operations, or unpredictable events such as cache misses.

The combination of these factors results in custom hardware exhibiting poor performance when implementing general-purpose sequential code via high-level synthesis. Venkatesh et al [17] generate custom hardware *conservation cores* from hot regions of sequential general-purpose code to improve energy efficiency. However, their cores are at best only able to match the performance of an in-order MIPS 24KE processor.

Matching Superscalar performance with the VSFG. To overcome the sequential performance issues of custom hardware, we propose two key changes during high-level synthesis:

- Instead of using a static scheduling based execution model for custom hardware, a dynamically scheduled execution model like *Spatial Computation* should be used [3], that implements code as a static dataflow graph in hardware.
- A new compiler IR is needed to replace the CDFG based IRs that are traditionally used for hardware synthesis. This new IR should be based on the Value State Dependence Graph (VSDG) [14] as it has no explicit representations of control-flow, instead only representing the necessary value and state dependences in the program.

A new compiler IR for implementing spatial computation, called the Value State Flow Graph (VSFG), has been developed based on the VSDG but modified for direct implementation in hardware as a static dataflow machine. The VSFG for Listing 1 is shown in Figure 1b. Unlike the CDFG, there is no subdivision of operations into basic blocks, and consequently, no notion of *flow of control* from one block to another. Instead, only dataflow dependences are represented, along with a sequentializing state-edge (dashed line in Figure 1b) that ensures that all side-effecting operations occur in the correct program order.

Instead of flow of control, the execution of operations is controlled through the use of *predicates* (purple dotted line in Figure 1b) – boolean expressions generated based on the control-flow of the CDFG. The VSFG is also a hierarchical graph – all loops and function calls are represented as nested subgraphs. From the perspective of any level in the graph hierarchy,

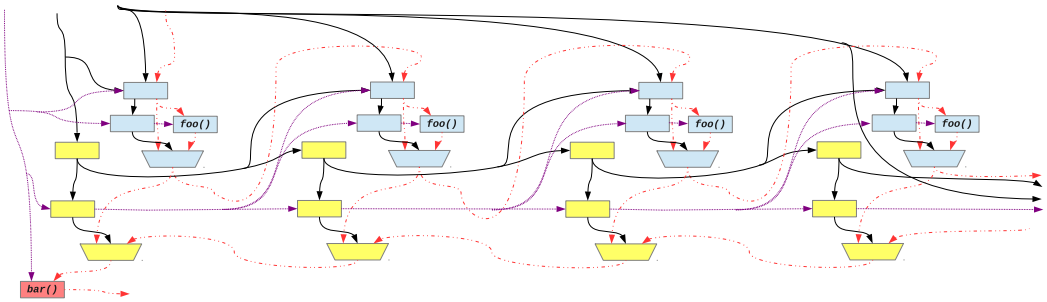


Figure 3 The VSFG from Figure 1b with the loop unrolled 4 times.

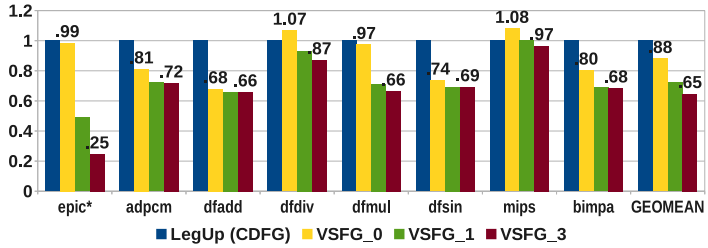
nested subgraphs appear as ordinary dataflow operations with their defined dataflow inputs and outputs (including predicate and state). The only difference being that nested subgraphs may exhibit a variable execution latency. As with other dataflow operations, multiple such subgraphs may execute concurrently, so long as their dataflow dependences are satisfied.

The VSFG is a directed acyclic graph, so loops are implemented without introducing explicit cycles in the graph by representing them instead as tail-recursive functions. As can be seen from Figure 1b, the *"Next iteration of the for-loop"* is represented as a nested subgraph in the same way as both the *foo()* and *bar()* functions. This presents a key advantage when we try to extract ILP from across multiple iterations of a loop. Any of the nested subgraphs in a VSFG can be *flattened* into the body of the parent graph. In the case of loops, flattening the subgraph representing the tail-recursive loop call is essentially equivalent to unrolling the loop. Figure 3 shows the for loop *unrolled* 4 times. Furthermore, as each loop is implemented within its own subgraph, this type of unrolling may be implemented within different subgraphs independently of others. Therefore, it is possible in the VSFG to exploit ILP by unrolling each loop within a loop nest independently of the other loops. (Note that in the actual hardware implementation, cycles must be reintroduced once the appropriate degree of unrolling has been done for each loop).

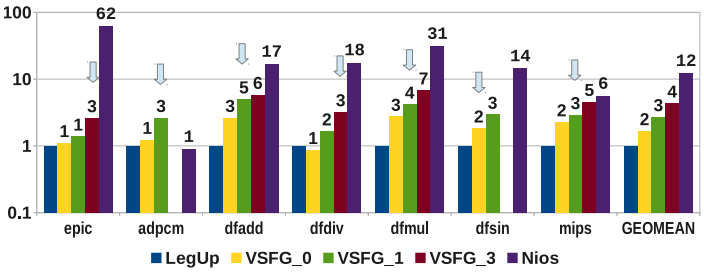
Thanks to the explicit control predicates, is also possible to perform aggressive control flow speculation by selectively controlling the execution of subgraphs. For instance, for the *foo()* function subgraph, we may choose whether this function executes speculatively or not: the predicate input to *foo()* may be used to only allow its execution when the predicate is true, thereby providing no speculation. Alternatively, the function may start executing irrespective of the predicate value. In this case, the predicate value will be passed into the subgraph, where side-effect free operations may execute speculatively even before the predicate value becomes available (all side-effecting operations will always be predicated).

Another advantage of having control flow converted in to boolean predicate expressions is the ability to perform *control dependence analysis* to identify regions of code that are control-flow equivalent and may therefore execute in parallel (provided all state and dataflow dependences are satisfied). Consider the *bar()* function in the CDFG (Figure 1a). Despite the aggressive branch prediction, a superscalar processor will not be able to start executing the *bar()* function until it has exited the loop. Similarly, when the *if* branch is predicted-taken, the superscalar processor must switch from executing multiple dynamically unrolled copies of the for-loop and instead focus on executing the control-flow within *foo()*. This is because a conventional processor can only execute along a *single flow of control* [12].

The VSFG on the other hand can use control dependence analysis to identify the control-flow equivalence between the contents of the for-loop and the *bar()* function, and so long as the dataflow and state-ordering dependences are resolved, the *bar()* function may start



(a) Performance in Cycle Counts vs LegUp.



(b) Energy Consumption comparison vs LegUp and an Altera Nios IIf in-order Processor.

Figure 4 Performance and Energy Comparison of VSFG, normalized to LegUp results.

executing in parallel with the for-loop. Similarly, the contents of the *foo()* subgraph can also execute in parallel with its parent graph. If we combine this concurrency with loop unrolling, it becomes possible in Figure 3 to execute multiple copies of the loop and *foo()*, in parallel with the execution of *bar()*! This ability to execute multiple regions of code concurrently is equivalent to enabling execution along *multiple flows of control*, and can potentially expose greater ILP than even a superscalar processor.

3 Evaluation Methodology & Results

To evaluate our IR and execution model, we implemented a HLS toolchain to compile LLVM IR to custom hardware. We present results for 3 versions of our VSFG hardware: VSFG_0 has no loop unrolling/flattening, VSFG_1 has all loops unrolled once, and VSFG_3 has all loops unrolled thrice. We compare our results against LegUp [4], an established HLS tool that utilizes the CDFG IR and static scheduling (Figure 4). The input LLVM IR to both toolchains was optimized with '-O2' flags, and with no link-time inlining or optimization. All generated circuits were targeted for implementation on an Altera Stratix IV GX FPGA.

We also compare performance against two conventional processors: an Intel Nehalem Core i7 Processor – simulated using Sniper from Intel [5] – as well as an Altera Nios IIf processor, also implemented on the Altera Stratix IV FPGA (Figure 5). Both processors are configured with perfect L1 caches and a hit latency of 1 cycle. Six of the benchmarks used are part of the CHStone HLS benchmark suite [10], while two other are home grown: *bimpa* is a neural network simulator, and *epic* is a matrix transpose function – both of the latter wer selected specifically because they have complex and data-dependent control flow.

Our generated hardware achieves a geometric mean speedup of 1.55× (max 4.05×) over equivalent hardware generated by LegUp (Figure 4a), and is able to better approach (in some cases even exceed) the performance of the Intel Core i7 (Figure 5). While this performance

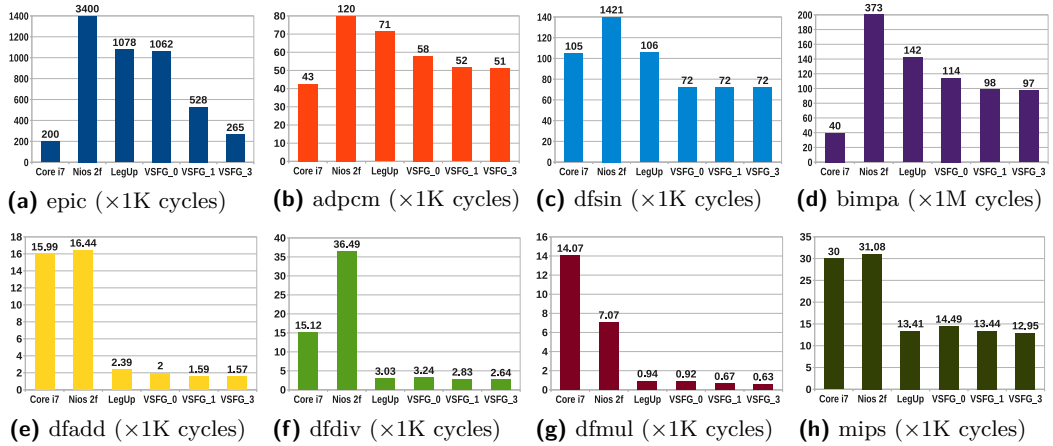


Figure 5 Performance Comparison (Cycle Count) vs an out-of-order Intel Nehalem Core i7 processor, and an Alteral Nios IIf in-order processor.

incurs an average 3× higher energy cost than LegUp, the VSFG-based hardware's energy dissipation is still only 0.25× that of a highly optimized in-order Nios IIf processor (Figure 4b).

4 Conclusion

We combine static-dataflow execution model with a new compiler IR in order to match the sequential performance of superscalar processors in custom hardware. The hierarchical and control-flow agnostic nature of the VSFG not only enables the exploitation of ILP from across multiple levels of a loop nest, but also enables control-dependence analysis and allows execution along multiple flows of control, potentially exploiting more ILP than is theoretically possible for even superscalar processors.

Our results show a performance improvement of as much as 35% over LegUp, at a 3× higher average energy cost. This performance is close to what is achieved by an Intel Nehalem Core i7, while being only 0.25× the energy cost of an in-order Altera Nios IIf processor. We believe that our new IR, coupled with the Spatial Computation execution model is a promising step towards addressing the problem of dark silicon by facilitating the development of high-performance custom hardware.

References

1 Geoffrey Blake, Ronald G. Dreslinski, Trevor Mudge, and Krisztián Flautner. Evolution of thread-level parallelism in desktop applications. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 302–313, New York, NY, USA, 2010. ACM.

2 M. Budiu, P.V. Artigas, and S.C. Goldstein. Dataflow: A complement to superscalar. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 177 –186, march 2005.

3 Mihai Budiu, Girish Venkataramani, Tiberiu Chelcea, and Seth Copen Goldstein. Spatial computation. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XI, pages 14–26, New York, NY, USA, 2004. ACM.

4 Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-

based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 33–36, New York, NY, USA, 2011. ACM.

5 Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2011.

6 Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

7 N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor. The greendroid mobile application processor: An architecture for silicon's dark future. *Micro, IEEE*, pages 86–95, March 2011.

8 Ed Grochowski and Murali Annavaram. Energy per instruction trends in intel® microprocessors. 2006.

9 Sumit Gupta, Nikil Dutt, Rajesh Gupta, and Alexandru Nicolau. Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, DATE '04, pages 10114–, Washington, DC, USA, 2004. IEEE Computer Society.

10 Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. Chstone: A benchmark program suite for practical c-based high-level synthesis. In *ISCAS'08*, pages 1192–1195, 2008.

11 Srikanth Kurra, Neeraj Kumar Singh, and Preeti Ranjan Panda. The impact of loop unrolling on controller delay in high level synthesis. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 391–396, San Jose, CA, USA, 2007. EDA Consortium.

12 Monica S. Lam and Robert P. Wilson. Limits of control flow on parallelism. In *Proceedings of the 19th annual international symposium on Computer architecture*, ISCA '92, pages 46–57, New York, NY, USA, 1992. ACM.

13 Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.

14 Alan C. Lawrence. Optimizing compilation with the Value State Dependence Graph. Technical Report UCAM-CL-TR-705, University of Cambridge, Computer Laboratory, December 2007.

15 Daniel S. McFarlin, Charles Tucker, and Craig Zilles. Discerning the dominant out-of-order performance advantage: is it speculation or dynamism? In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '13, pages 241–252, New York, NY, USA, 2013. ACM.

16 R. Namballa, N. Ranganathan, and A. Ejnioui. Control and data flow graph extraction for high-level synthesis. In *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*, pages 187–192, 2004.

17 G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M.B. Taylor. Conservation cores: reducing the energy of mature computations. *ACM SIGARCH Computer Architecture News*, 38(1):205–218, 2010.

A Graph based approach for Co-scheduling jobs on Multi-core computers

Huanzhou Zhu and Ligang He

University of Warwick
Coventry, UK
{zhz44, liganghe}@dcs.warwick.ac.uk

Abstract

In a multicore processor system, running multiple applications on different cores in the same chip could cause resource contention, which leads to performance degradation. Recent studies have shown that job co-scheduling can effectively reduce the contention. However, most existing co-schedulers do not aim to find the optimal co-scheduling solution. It is very useful to know the optimal co-scheduling performance so that the system and scheduler designers can know how much room there is for further performance improvement. Moreover, most co-schedulers only consider serial jobs, and do not take parallel jobs into account. This paper aims to tackle the above issues. In this paper, we first present a new approach to modelling the problem of co-scheduling both parallel and serial jobs. Further, a method is developed to find the optimal co-scheduling solutions. The simulation results show that compare to the method that only considers serial jobs, our developed method to co-schedule parallel jobs can improve the performance by 31% on average.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Co-scheduling algorithm, Multicore processor, Cache interference, Parallel Job

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.144

1 Introduction

Multicore processors have now become a mainstream product in CPU industry. In a multicore processor, multiple cores reside on the same chip and share the resources in the chip. However, running multiple applications on different cores in the same chip could cause resource contention, which leads to performance degradation. Many research studies have shown that it is possible to isolate some resources, such as disk bandwidth [15], network bandwidth [8] for the co-running jobs. However, it is very difficult to isolate the on-chip last level cache (LLC). This problem is known as the shared cache contention and has been studied in literature [9,11,18]. The existing approaches to addressing on-chip shared cache contention fall into the following three categories: 1) Architecture-level solutions that focus on improving the hardware to provide isolation among threads [13] [14], 2) System-level solutions that focus on partitioning the cache for each application [16] [12], and 3) Software-level solutions that tend to develop the contention-aware scheduler to reduce the contention [5] [7]. In the above three categories, the architecture-level solution is still under active development by the processor vendors. The cache partitioning solution requires many changes in the existing system-level software (such as operating system), and therefore incurs high implementation cost. The third approach, the contention-aware schedulers, is a fairly lightweight approach, and therefore attracts many researchers' attention, which is also the focus of this work.

2013 Imperial College Computing Student Workshop (ICCSW'13).
Editors: Andrew V. Jone, Nicholas Ng; pp. 144–151

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A number of contention-aware co-schedulers have been developed in the literature [1, 4, 7]. These studies demonstrated that contention-aware schedulers can deliver better performance than the conventional schedulers. However, they do not aim to find the optimal co-scheduling performance. It is very useful to know the optimal co-scheduling performance. With the optimal performance, the system and co-scheduler designers can know how much room there is for further performance improvement. In addition, knowing the distance between current performance and optimal performance can help the scheduler designers to make the tradeoff between scheduling efficiency and scheduling quality.

The co-schedulers discussed in literature only consider serial jobs (each of which runs on a single core), and do not take parallel jobs into account. However, both parallel jobs and serial jobs often exist in a multicore computer system. For example, both parallel and serial jobs are submitted to a cluster consisting of multi-core computers.

The work in [9] modelled the optimal co-scheduling problem for serial jobs as an integer programming problem. However, we will show in this paper (Section 2) that this modelling approach cannot be extended to parallel jobs. This motivates us to develop a new method that is flexible to model the problem of co-scheduling both serial and parallel jobs.

In this paper, we first present a new approach to modelling the problem of co-scheduling both parallel and serial jobs. Further, a method is developed to find the optimal co-scheduling solutions.

We have conducted the simulation experiments to evaluate the co-scheduling algorithms we developed. The results show that taking parallelism into account can significantly improve performance. More specifically, if the method developed for serial jobs is used to co-schedule a mix of parallel and serial jobs, the performance achieved by the new method that takes parallel jobs into account is 31% better on average than only considering serial jobs.

The rest of the paper is organized as follows. Section 2 formalizes the problem of co-scheduling a mix of parallel and serial jobs. Section 3 presents a method to find the optimal co-scheduling solutions. The experimental results are presented in Section 4. Finally, the paper is concluded in Section 6.

2 Formalizing the problem of co-scheduling parallel jobs

The work in [9] i) formalized the problem of co-scheduling serial jobs, and ii) proposed an approach to modelling and finding the optimal co-scheduling solution. In this section, we first briefly summarize their formalization method in Subsection 2.1, then in Subsection 2.2 extend the method to incorporate parallel jobs, and present our own approach to modelling the problem of co-scheduling a mix of parallel and serial jobs, and developing the methods to solve the model for optimal co-scheduling solutions.

2.1 Formalizing the problem of co-scheduling serial jobs [9]

The work in [9] shows that on a multicore processor, the co-running jobs are generally slower than when they run alone due to resource contention. This performance degradation is called co-run degradation. The co-run degradation of a job is defined as the difference between the execution time of the job when it co-runs with a set of other jobs and its execution time when it runs alone. Formally, the performance degradation of a job i is expressed in Eq. 1, where ft_i is the execution time when job i runs alone, S is a set of jobs and $ft_{i,S}$ is the

execution time when job i co-runs with the set of jobs in S.

$$D_i = \frac{ft_{i,S} - ft_i}{ft_i} \tag{1}$$

In the co-scheduling problem, n jobs need to be allocated to a cluster of u-core multiprocessors so that each core is allocated with one job. m denotes the number of u-core multiprocessors needed, which can be calculated as $\lceil \frac{n}{u} \rceil$. The objective of the co-scheduling problem is to find the optimal way to partition n jobs into m u-cardinality sets, so that the sum of D_i in Eq. 1 over all n jobs is minimized, which is shown in Eq. 2. Note that if n is not the multiple of u, i.e., $n\%u \neq 0$, we can simply generate $u - n\%u$ imaginary jobs whose performance degradation with any job is 0.

$$min \sum_{i=1}^{|n|} D_i \tag{2}$$

2.2 Formalizing the problem of co-scheduling parallel jobs

As defined in Eq. 2, in order to find the optimal co-scheduling solution, the objective is to minimize the sum of the performance degradation experienced by each job. This objective function is designed for co-scheduling serial jobs. A parallel job consists of multiple processes (or threads). Applying Eq.2 directly to the case involving parallel jobs, the total degradation of a mix of parallel and serial jobs is the sum of the degradation experienced by each process in all parallel jobs plus the sum of the degradation by each serial job. However, the finishing time of a parallel job is determined by its slowest process. A larger degradation for a process indicates a longer execution time for that process. Therefore, no matter how small the degradation is for other processes, they have to wait until the process with the largest degradation finishes, which essentially means that all processes suffer the same degradation as the largest degradation. Thus, the total degradation for a parallel job is the largest degradation among all degradations experienced by its processes multiplied by the number of the processes in the parallel job, which can be formally defined as $M_i \times max(D_{ij})$, where M_i is the number of processes in parallel job i and D_{ij} is the degradation of process j in parallel job i. Based on this analysis, we re-formulate the objective function of finding the optimal co-scheduling solution for a set of jobs containing parallel jobs. The objective function is expressed in Eq. 3, where J is a set of all jobs, PJ is the set of parallel jobs in J, $|J|$ and $|PJ|$ represent the number of the jobs in the set J and PJ, respectively.

$$min(\sum_{i=1}^{|PJ|}(M_i \times max\{D_{ij}\}) + \sum_{i=1}^{|J|-|PJ|} D_i) \tag{3}$$

In order to solve this problem, we propose a new method to model the problem of co-scheduling serial jobs. The new modelling approach is flexible and can be extended to incorporate parallel jobs.

3 Modelling the co-scheduling problem

3.1 Graph Model of the problem

As formalized in Section 2, the objective of solving the co-scheduling problem is to find a way to partition n jobs, j_1, j_2, ..., j_n, into m u-cardinality sets, so that the total performance

degradation of all jobs is minimized. The number of all possible u-cardinality sets is $\binom{n}{u}$. In this paper, a graph called the co-scheduling graph is constructed, to model the co-scheduling problem. There are $\binom{n}{u}$ nodes in the graph and a node corresponds to a u-cardinality set. The ID of a node is coded a u-digit number, using the IDs of the jobs in the corresponding u-cardinality set. In the encoding, the job IDs are always placed in an ascending order from the most to the least significant digit. The weight of a node is defined as the total performance degradation of the u jobs in the node. The nodes are organized into different levels in the graph. The i-th level contains all nodes whose first digit of the ID is i. In each level, the nodes are placed in the ascending order of their ID's. A *start* node and an *end* node are added as the first level (level 0) and the last level of the graph, respectively. The weights of the start and the end nodes are both 0. Figure 1 illustrates when co-scheduling 6 jobs to 2-core processors, how to code the nodes in the graph, and how to organize the nodes into different levels. Note that for the clarity of the figure we did not draw all edges.

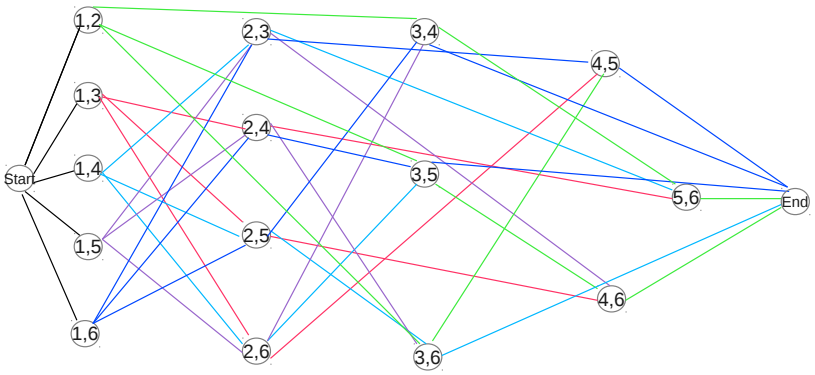


Figure 1 Degradation graph for 6 jobs on dual core system, the number in each node represents a Job ID, and edges with same color forms a group of possible schedule.

3.2 Optimal Parallel aware Shortest Path algorithm

A path from the start to the end node in the graph forms a co-scheduling solution if the path does not contain duplicated jobs, which is called a valid path. The distance of a path is defined as the sum of the weights of all nodes on the path. Finding the optimal co-scheduling solution is equivalent to finding the shortest valid path from the start node to the end node. In this paper, a algorithm, called OP-SCG (Optimal Parallel aware Shortest path algorithm for the Co-scheduling Graph) is developed to find the shortest valid path in the constructed graph. OP-SCG is adapted from Dijkstra's shortest path algorithm [3]. The main differences between OP-SCG and Dijkstra's algorithm lie in three aspects: 1) there are no edges between nodes in the graph in OP-SCG, and the edges are established as the algorithm progresses, 2) the invalid paths, which contain the duplicated jobs, have to be disregarded, and 3) the ability to compute degradation of parallel jobs.

In this algorithm, every node v of the graph contains some attributes, in which the $v.distance$ attribute records the length of the shortest path from the start node to node v, the $v.path$ attribute is a list and records the sequence of nodes in the shortest path up to v. The purpose of this design is to avoid spending time checking whether adding a new node will invalidate the resulting path.

In Algorithm 1, object Q is a list that holds jobs in ascending order of all paths that have been visited by the algorithm. For example, if the algorithm visited the paths [(1,3),(2,4)] and

Algorithm 1 The OP-SCG Algorithm

```
1  for every node v in the graph do
2      v.distance = 0;
3      v.previous = NULL;
4      Q = start.ID;
5      v = start;
6      while v != end
7          for every level l from v.level + 1 to end.level do
8              if l is not a digit of the ID of the nodes in v.path
9                  valid_l = l;
10         for every node k in the valid_l level do
11             if the jobs in node k are not in v.path:
12                 if k contains parallel job:
13                     label = compute_label(k, v + v.previous);
14                 else:
15                     label = v.distance + k.weight;
16             if k + v.path is not in Q or label < Q[k + v.path]:
17                 Q[k + v.path] = label;
18                 k.path = v.path + k;
19     remove node v from Q;
20     obtain such a node v from Q that has the smallest v.distance;
```

[(1,3),(2,5)], the data stored in Q is [(1234),(1235)]. For every node it visits, the algorithm searches for a valid level (Line 7-9), which is a level that contains at least one node that can form a valid path with the nodes in the current partial path. After a valid level is found, Algorithm 1 continues to search this level for the valid nodes that can form a valid path (Line 10-11). If a valid node, k, is found, the algorithm further checks if there are any parallel jobs in this node, and calculates the distance with function *compute_label* if the node k contains parallel jobs. Otherwise, the distance is computed by adding the previous distance with weight of k (Line 12). If node k has not been visited yet or the new path will form a shorter path (Line 16), the algorithm either adds node k to Q or updates the distance stored in $Q[k]$ (Line 16-18).

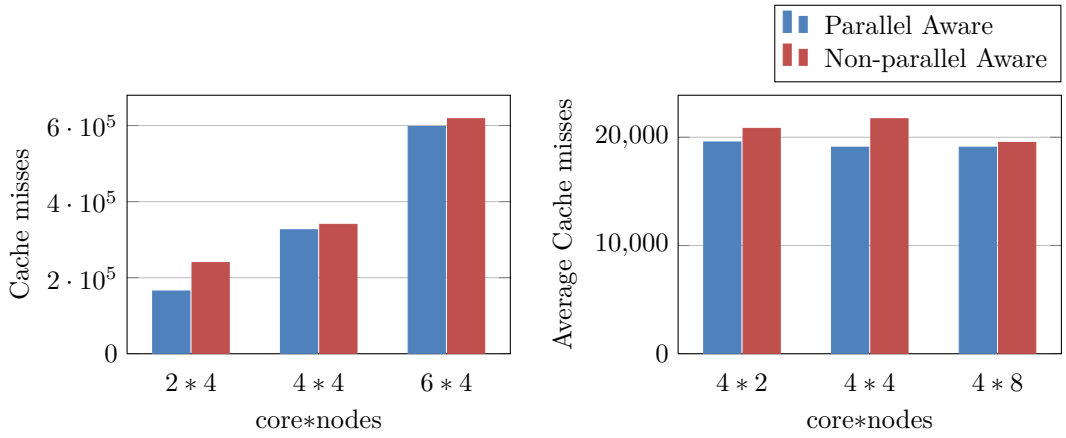
Also, by removing Line 12-14 from Algorithm 1, we obtain another algorithm, called B-SCG (Basic Shortest path algorithm for the Co-scheduling Graph) which is used to find the shortest path in the graph when there are only serial jobs.

4 Evaluation

In this section, we present the scheduling results produced by our optimal algorithm through a set of simulation tests. The configuration of simulation environment is based on Intel Core 2 series processor with 2 cores, 4 cores, 6 cores respectively, all running 2.66GHz, and having one shared 6MB, 4096 sets*24 way set associative last level cache.

4.1 Co-scheduling Result

In order to examine the capability of OP-SCG algorithm, we compare the results produced by B-SCG algorithm with it. The simulation is conducted by scheduling a set of artificially generated jobs. The degradation value of every node in the co-scheduling graph is computed by prediction model developed by Dhruba et.al [2]. As required by Dhruba's method, a stack distance profile with randomly generated cache hits and misses is assigned to each job, since we are interested in difference between results produced by two algorithms, the randomness has negligible influence on this experiment since both algorithms will operate on same job set.



(a) (b)

Figure 2 Changing core number and node number.

The first simulation tested the correctness of mathematical model and OP-SCG algorithm. This simulation was conducted by scheduling 8, 16, 24 jobs to 4 nodes cluster with 2, 4, and 6 cores respectively, half of which were parallel processes. The result is shown in Figure 2a. The metric used in this experiment is the total cache misses of each job. It is worthwhile to note that lower cache misses suggests lower performance degradation.

As shown in the Figure 2a, the parallel aware optimal scheduler produces lower cache misses in all cases due to consideration of parallelism. The average distance between the two algorithms is 31%. This result not only demonstrates the correctness of the algorithm presented in this paper but also proves that considering parallelism can significantly improve performance. In addition, the result also shows that the cache misses increase as the core number increases. Because the cache size does not increase as core number increases, the degradation increases since there are more jobs competing for the limited number of cache lines.

Since the basic principle of co-scheduling algorithm is to balance the on-chip shared resource usage, the second experiment was conducted to examine this ability of algorithm presented in this paper. In this experiment, 8, 16, 32 jobs with 50% parallel processes were scheduled to clusters with 2, 4, and 8 nodes respectively, each node has a quad-core processor. The results are shown in Figure 2b.

The metric used in this experiment is average cache misses of all jobs, the parallel aware optimal scheduler produced the lowest average cache misses among three cases again. The average cache misses between three cases are very similar, the difference between the highest and lowest result being 3%. This result suggests that the algorithm balanced the resource usage among every node within the cluster well, which also means that the fairness of this algorithm is good.

5 Discussion

The primary goal of this paper is to provide the theoretical insight for finding optimal co-scheduling with parallel job considered. Apart from that, practical co-scheduler designs can benefit from this work in two ways: Firstly, the optimal model presented in this paper provided a sufficient way of evaluating the co-scheduling results when parallel jobs are

considered. Knowing the optimal solution is important for practical co-schedule system design, because the tradeoff between efficiency and quality can be made based on knowledge about the distance between current results and the optimal solution. Secondly, the algorithm proposed in this work can be directly used in a proactive co-scheduling system. Predicting the co-run performance has been widely studied by many researchers (e.g., [2] [6] [10] [17]). Those studies make it possible to predict co-run performance accurate and efficiently. With accurate prediction, the proactive schedulers may use the algorithm proposed in this work to determine the optimal or near-optimal schedules.

In this work, we assumed that each core will execute only single job, however, the effectiveness of our approach is not strictly limited by this assumption. If there are multiple jobs running on same core, they are more likely to be executed in a time-sharing basis, therefore, our algorithm will still be an essential component for finding the optimal schedule in each time slice. The scheduler can re-schedule jobs at time slice boundaries.

6 Conclusion

This paper explored the problem of parallel aware optimal job co-scheduling on multiprocessor system. The paper built a mathematical model that can be used to find the optimal scheduling with consideration of parallel jobs. Based on this model the paper described an optimal parallel aware co-scheduling algorithm (OP-SCG) for multicore processor systems by formulating the problem as a shortest path problem. The experiment results show that by considering parallelism, the parallel aware optimal algorithm decreases the average performance degradation by 31%.

The mathematical model and algorithm in this paper offer the theoretical and practical support for the evaluation of co-scheduling systems.

References

1 Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. Contention-aware scheduling on multicore systems. *ACM Transactions on Computer Systems (TOCS)*, 28(4):8, 2010.

2 Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, HPCA '05, pages 340–351, Washington, DC, USA, 2005. IEEE Computer Society.

3 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2001.

4 Alexandra Fedorova, Sergey Blagodurov, and Sergey Zhuravlev. Managing contention for shared resources on multicore processors. *Communications of the ACM*, 53(2):49–57, 2010.

5 Alexandra Fedorova, Margo Seltzer, and Michael D Smith. Cache-fair thread scheduling for multicore processors. *Division of Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-17-06*, 2006.

6 Alexandra Fedorova, Margo Seltzer, and Michael D Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 25–38. IEEE Computer Society, 2007.

7 J. Feliu, S. Petit, J. Sahuquillo, and J. Duato. Cache-hierarchy contention aware scheduling in cmps. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2013.

8 Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In *Middleware 2006*, pages 342–362. Springer, 2006.

9 Yunlian Jiang, Kai Tian, Xipeng Shen, Jinghe Zhang, Jie Chen, and Rahul Tripathi. The complexity of optimal job co-scheduling on chip multiprocessors and heuristics-based solutions. *Parallel and Distributed Systems, IEEE Transactions on*, 22(7):1192–1205, 2011.

10 Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122. IEEE Computer Society, 2004.

11 Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209. IEEE, 2007.

12 Min Lee and Karsten Schwan. Region scheduling: efficiently using the cache architectures via page-level affinity. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 451–462. ACM, 2012.

13 Kyle J Nesbit, James Laudon, and James E Smith. Virtual private caches. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 57–68. ACM, 2007.

14 Shekhar Srikantaiah, Mahmut Kandemir, and Mary Jane Irwin. Adaptive set pinning: managing shared caches in chip multiprocessors. *ACM Sigplan Notices*, 43(3):135–144, 2008.

15 Jianyong Zhang, Anand Sivasubramaniam, Qian Wang, Alma Riska, and Erik Riedel. Storage performance virtualization via throughput and latency control. *ACM Transactions on Storage (TOS)*, 2(3):283–308, 2006.

16 Xiao Zhang, Sandhya Dwarkadas, and Kai Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 89–102. ACM, 2009.

17 Qin Zhao, David Koh, Syed Raza, Derek Bruening, Weng-Fai Wong, and Saman Amarasinghe. Dynamic cache contention detection in multi-threaded applications. *ACM SIGPLAN Notices*, 46(7):27–38, 2011.

18 Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 129–142. ACM, 2010.